Out of Office (OoO) Language Specification

Prairie Resch

April 30, 2024

Note: some parts of this document might already be outdated with respect to the code I've written. Everything keeps changing.

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 (how exactly DOES one clean a 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

1.
```
define date range (04/17/2023, 05/18/2023)
new location Home
new Dog Calvin
new Dog Alma
new action (walk Alma) 3 Day
new action (walk Calvin) 5 Day
new action (feed both dogs) 2 Day
new action wash 1 Week
```

This program creates a schedule for between the dates April 17, 2024 to May 18, 2024. The schedule introduces a dog-sitter to Alma and Calvin, and instructs the sitter to perform walk Alma three times a day, walk Calvin 5 times a day, feed both dogs twice 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. I'm still not sure how/if I can connect objects with actions.

2.
```
define date range (05/04/2045, 08/06/2046)
new Plant pothos
new action repot 2 Year
```

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.
```
new Child Dave
new Child (Bobkin VanHorn)
new Child Snim
new Child HotShot
new Child (Sunny Jim)
new Child (Zanzabar Buck Buck Mc Fate)
new action (come into the house, Dave) 1 Hour
```

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 the date of creation and an end date of one week later, since none have been given–the final pdf will note this) 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” once an hour, whatever that means. OoO might not yet be user-friendly enough for Mrs. McCave.

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. Additionally, the user doesn't need to know much about combinations–most of that will go on under the hood, I think.

Formal Syntax

```
<timeframe> ::= <date> * <date>
<date>      ::= int/int/int

<expr>      ::= new <new_thing> <expr>
             |  define date range <timeframe> <expr>
             |  assign <action> <instance> <expr>
             |  <num>
             |  <name>
             |  <empty>

<new_thing> ::= <type>
             |  <instance>
             |  <action>
             |  <location>

<type>      ::= string
<num>       ::= int
<name>      ::= string
<instance>  ::= <type> <name> [<action>]
<timescale> ::= Minute | Hour | Day | Week | Month | Year
<location>  ::= string * <instance> list
```

I'm still trying to figure out how/if to define type as something more than a string. At the moment, my code also has a Name and a Num type, which I've included here for that exact reason, but I'm not positive they will stay.

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.