Outline For Esther

1 Purpose

So here is what Esther should do:

1. Launch programs (e.g. open file f with program x).
2. Basic file management (i.e. copy file f to location ℓ.)
3. Free form input (e.g. with text t, email t to address a; here, both t and a are typed in by the user[1].)

Some design goals:

1. Be *fully and easily extensible*—see sections below.
2. No "hacking around" in the source code; i.e. be well-designed.
3. Less than 500 lines of code for the main program.
4. The GUI code should be separated from the "backend" code.

The rest of this document will talk about possible implementations.

2 Abstract Data Structures

The Program. The actual program Esther consists is a tuple $(S, \mathcal{P}, f, k)$ where

- S is a state such that each $P_i \in \mathcal{P}$ (see below).
- $\mathcal{P}$ is a collection of plugins (see below).
- f is a function,
$$f : \{1, \ldots, n\} \times \mathcal{O} \mapsto \mathcal{P}^n \tag{1}$$
where $\mathcal{O}$ is the set of all objects (see below), and n is the "size" of the state (see below).
- k is a function, $k : \mathcal{K} \times S \to S$, where K is the set of all keyboard strokes.

States. A state $S = (\mathcal{B}, B)$ is a tuple where

- $\mathcal{B} = \{(B_1, P_1), \ldots, (B_n, P_n)\}$ is a set consisting of tuples of boxes and plugins.
- B is a box such that $B = B_i$ for some $(B_i, P_i) \in \mathcal{B}$.

Boxes. A box $B = (D, E, \mathcal{L}, S, \delta, \epsilon, \lambda, f)$ is a tuple where:

[1] Actually, I suppose that a can be autocompleted when Esther can read from the address book.

- D is a string (it is displayed to the user; *display*).
- E is a string (it is what the user types in; *entry*).
- $\mathcal{L}$ is a *list* of strings (it is displayed to the user in a drop-down list).
- S is an **object**; it represents the currently selected object in the box.
- δ is a handle[2] to a GUI element that corresponds to D; it displays D to the user.
- ϵ is a handle to a GUI element that displays E to the user. (It should be a textbox).
- λ is a handle to a GUI element that displays L to the user. (It should be a list menu.)
- f is one member of the set $\{0, 1, 2\}$. It equals 0 if δ has focus; it equals 1 if ϵ has focus; it equals 2 if λ has focus.

Objects. Generally speaking, there are two kinds of objects, **Ingredients** and **Recipes**.

Ingredients. An ingredient $I = (\mathcal{T}, r, \mathcal{D})$ is a tuple where:

- $\mathcal{T}$ is a set of strings; when $t \in \mathcal{T}$, I is said to be of type t. Thus an ingredient may have multiple types.
- r is a string; it is the (textual) *representation* of I.
- $\mathcal{D}$ is a dictionary, in the programming sense of the word. The keys in $\mathcal{D}$ are a function of t.

Recipes. A recipe is a tuple $R = (t, r, \mathcal{I}, f, \mathcal{D})$ where

- t is a string; the *type* or R.
- r is a string; it is the (textual) *representation* of R.
- $\mathcal{I} = \{t_1, \ldots, t_n\}$ is a list of strings. $\mathcal{I}$ is a function of t.
- f is a function of ingredients $I_1, \ldots, I_n$, such that I_k is of type t_k; f performs some task depending on its arguments.
- $\mathcal{D}$ is a dictionary whose set of keys is a function of t.

Plugins. A plugin $P = (t, S, D, K, H, f, g, d)$ is a tuple where

- t is a string, the *type* of the plugin.
- S is a list, consisting of any **objects** (see above); it is the *source* of the plugin.
- D is a dictionary, whose keys are a function of t.
- K is a set of keyboard strokes, e.g. $K = \{0 - 0, A - Z, a - z, \texttt{C} - \texttt{a}, \texttt{C} - \texttt{e}\}$.
- H is another set of keyboard strokes.

[2] Any programming construct that allows access to the GUI element.

- f is a function
$$f : K \times \mathfrak{B} \to \mathfrak{B} \tag{2}$$
where $\mathfrak{B}$ is the set of all boxes.
- g is a function,
$$g : H \times S \to S. \tag{3}$$
- d is a funtion,
$$d : \mathfrak{B} \to \mathfrak{B}. \tag{4}$$

It is understood that f, g, and d are defined in terms of the components of P.

3 Data Structure Descriptions

3.1 Boxes

Users see the interface of Esther as a series of **boxes** (see Quicksilver's interface). Appearance-wise, a box is a GUI-element consisting of three parts:

1. A non-editable frame in which text is displayed.
2. An editable text-field in which the user enters a query.
3. A drop down list that continually updates in response to user's query.

(In future versions, another GUI element will be added that can display an icon/picture of the user's selection.) These account for the variables δ, ϵ, and λ, respectively, that appear in the definition of a box above.

Any any given moment, the exact contents of these GUI elements is determined by

1. D—what is displayed in the non-editable frame.
2. E—what the user query is.
3. $\mathcal{L}$—a list of strings displayed in the drop down list. They are displaying the textual representations of objects matching the query (see next section).

The overarching rule is

The variables D, E, and $\mathcal{L}$ $\boxed{\text{always}}$ reflects the contents of GUI. In other words, these variables and the corresponding GUI elements change *simultaneously*[3].

The variable f designates which of the GUI elements has focus; therefore setting the focus means changing f, and conversely. Note

In addition to all of the above, each box keeps track of which object it is currently holding; this is the purpose of the component S. For example, box one could hold the object "Firefox", while the second box could hold the object "Run". To be proper, an object is not a string, but all objects have a name that is a string (this is explained below).

[3]In math lingo, this means "if and only if".

3.2 Objects

Objects are the central core of Esther. The nouns, verbs, objects of Quicksilver are all called **Objects** in Esther lingo. Here are some examples of objects:

- `Firefox`
- `http://www.google.com`
- `/home/user/my-document.doc`
- `/home/user/Music/song.mp3`
- `=3+5`
- `Run`
- `Open with`
- `Copy to`
- `Play in iTunes`
- `Pause DeadBeef`

One may think of Esther's purpose as an interface to select objects. As mentioned, objects come in two types:

1. Ingredients.
2. Recipes.

The naming is meant to reflect the idea that any action carried out by Esther is built from a (possibly-empty) set of ingredients and one (and only one) recipe. For example, the first 5 items above are ingredients, while the last 5 are recipes.

Put a different way: Ingredients by themselves don't do anything; they simply provide data. Recipes act on ingredients. Of course, some recipes, such as `Pause DeadBeef` don't require any ingredients.

All objects have a textual representation. For example, the string "Firefox" is not an object, but simply the textual representation of one. When the user enters a query, he is querying those objects whose textual representation matches the string he typed in. In reality, an object more data, as outlined in the previous section. They are elaborated on below.

3.2.1 Ingredients

Every ingredient have associated with it a set of types. For example "Firefox" could be associated with the types `Executable`, `File`, and `Browser`. The types of an ingredient is stored in the component $\mathcal{T}$.

The variable r stores the textual representation of the ingredient, as described above.

Finally, each ingredient stores some data about itself, and this data is stored in a dictionary called $\mathcal{D}$. The contents of the dictionary is decided by the type(s) of the Ingredient. For example, we *always* expect any ingredient with type `File` to have a `LOCATION` key, and so on.

3.2.2 Recipes

Recipes also have a type-set $\mathcal{T}$ and textual representation r.

As described, recipes take some ingredients as input and produces an "action". This action is encapsulated by the function f. It accepts an ingredient of type t_1, an ingredient of type t_2, and so on; the list of types is described in $\mathcal{T}$.

Like ingredients, recipes also have a dictionary $\mathcal{D}$ in which to store values, but its purpose is less clear-cut. It is there mainly for flexibility considerations.

4 Plugin Details

Plugins make the whole thing tick. To begin with, each box has associated with a plugin; which plugin a particular box is associated may change over time; the overlying function f described in Equation (1) is responsible for this. We will come back to this point later.

A plugin is principally responsible for the box it's attached to in three areas:

1. Initializing the boxes with default values.
2. Responding to keyboard input when the box has focus. This includes providing the box with a repertoire of objects on which a search of the user's query is done.
3. Coordinating with the other boxes.

All plugins possess a type t, and corresponding to this type, there is a dictionary $\mathcal{D}$ which stores whatever values needed for a plugin of type t.

Each time a plugin is loaded into a box, the function d of the plugin in called to initialize the box's GUI elements.

The repertoire described in item 2 above is called the "source" of the plugin, which is described by the variable S; it is a set of objects. For example, one possible source is "all executable files in the `/bin` folder" (of course, an object has rigidly defined components as above, but I hope the example is clear).

Each time a keystroke k is pressed in a box B, its corresponding plugin will call functions f and g, respectively, if $k \in K$ and $k \in H$. The first of these will update the box's GUI elements and internal data structures (e.g. reload drop down list in response to the new user query) using this repertoire. The second of these will refresh the repertoire.

Of course, both of these functions are free to modify the dictionary $\mathcal{D}$ for whatever purposes. Any keystroke not processed by these functions are sent back to the function k in the toplevel.

5 Program Flow