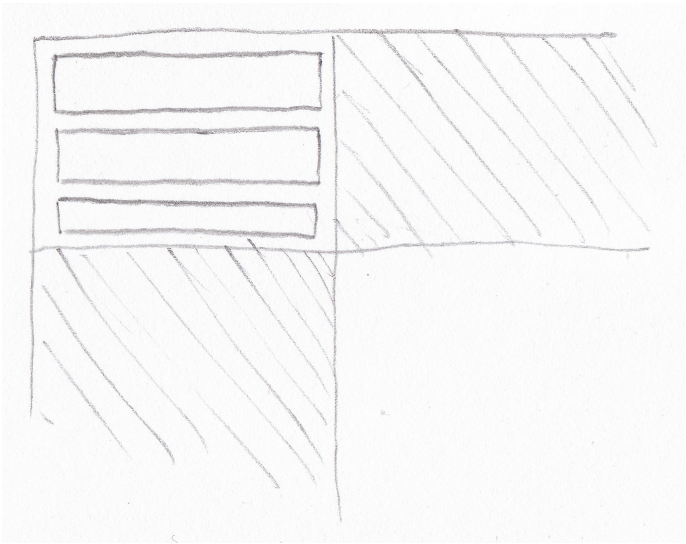
The Peyote User Interface

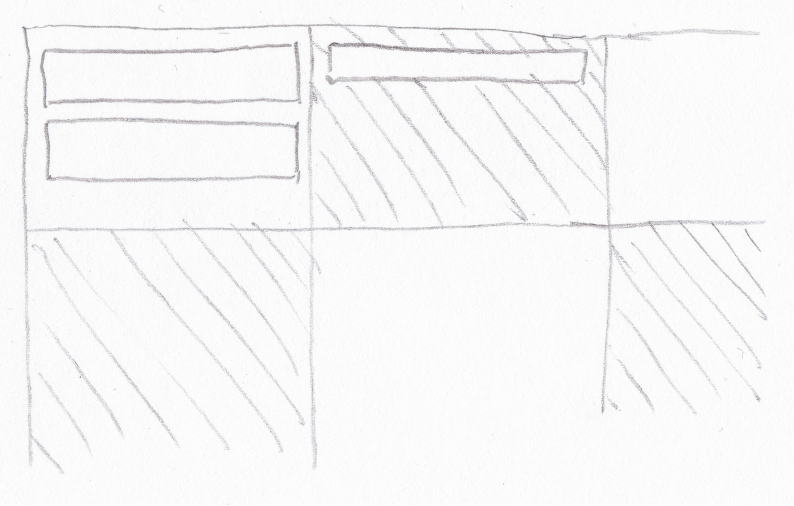
Tim Macfarlane

Glyphs

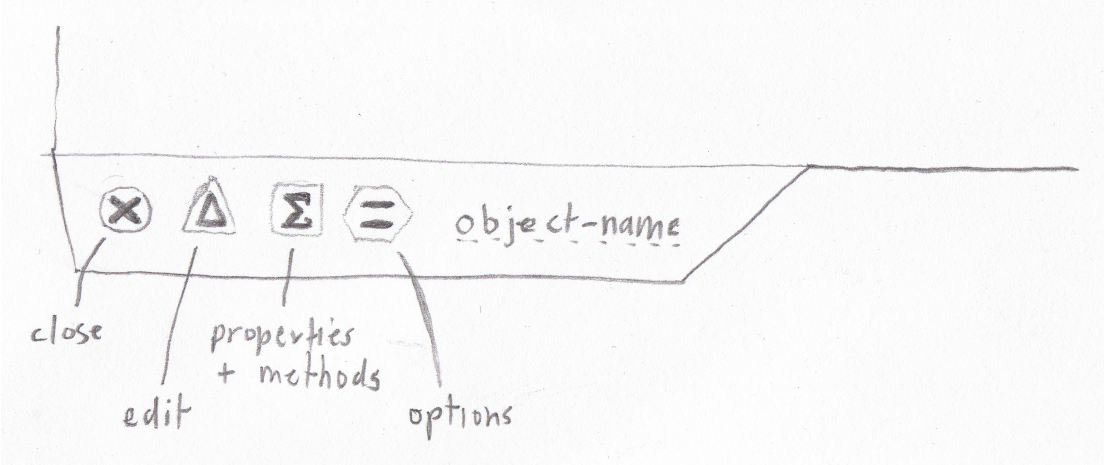
Glyphs are small visual elements that can both present objects as well as interact with them. The user can click buttons, drag sliders, enter text and explore complex datasets, anything that can be done in normal desktop software. Glyphs represent objects in the system; in fact several glyphs can represent the same object in the system, offering different views or merely the same view in different places. Glyphs contain everything that is necessary to view and interact with their object; they're mostly self-contained. They can be dragged and positioned around the larger user interface to suit the user's work.

Glyphs are usually stacked vertically, one on top of the other, in what is called glyph space. This is an area that is divided into squares, or cells. Each cell can contain any number of vertically stacked glyphs and glyphs can be moved from one cell to another. New cells are created adjacent to the ones that currently contain glyphs so you'll never run out of cells, the more cells you use, the more new cells will be available.

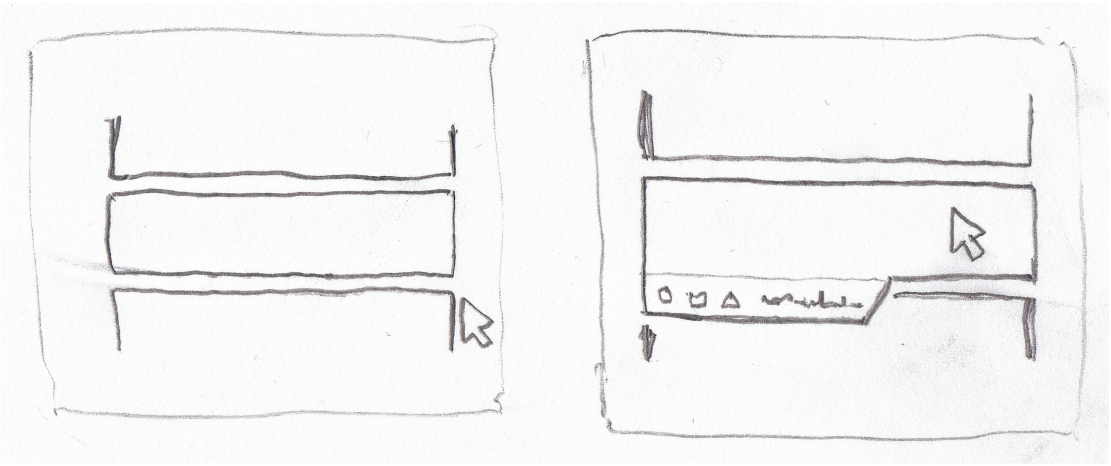




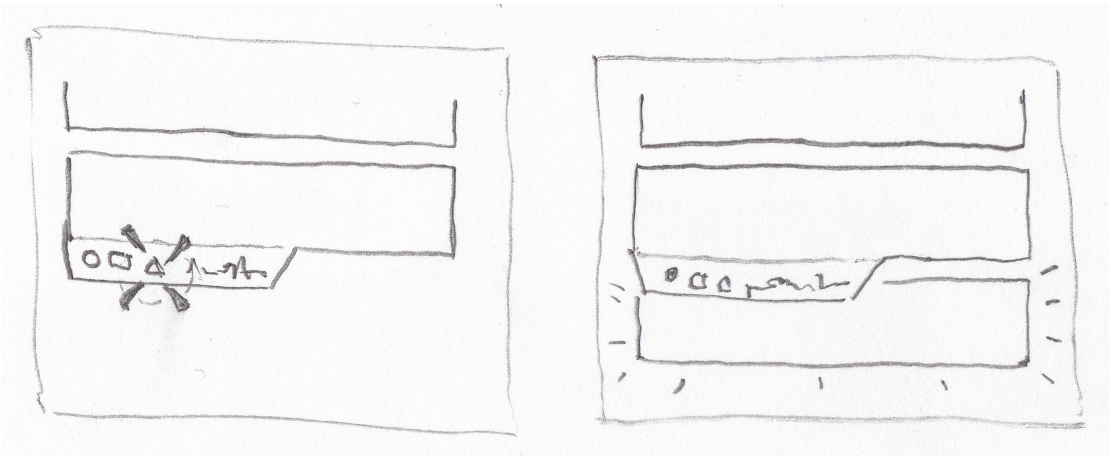
Glyphs are generally specialised to the object they're representing but all objects in the system will have common properties, and all glyphs can have common functionality. For this, glyphs are given **handles** that can be used to close the glyph, move, edit, inspect the underlying object or set display options. Since glyphs will naturally represent only certain aspects of an object, you can select a different glyph to view the same object in a different way.



The glyph can also be given a name, which is displayed and can be changed in the handle. This name can be used in formulas and computations, discussed later. The handle is not always visible, since handles can take up a lot of room on screen, especially if there are lots of glyphs visible. So a glyph's handle is only displayed if the mouse is hovering over the glyph.



Some glyphs, depending on how you use them, will create new objects in the system. When this happens, the new objects will have new glyphs placed for them underneath the glyph that created them.

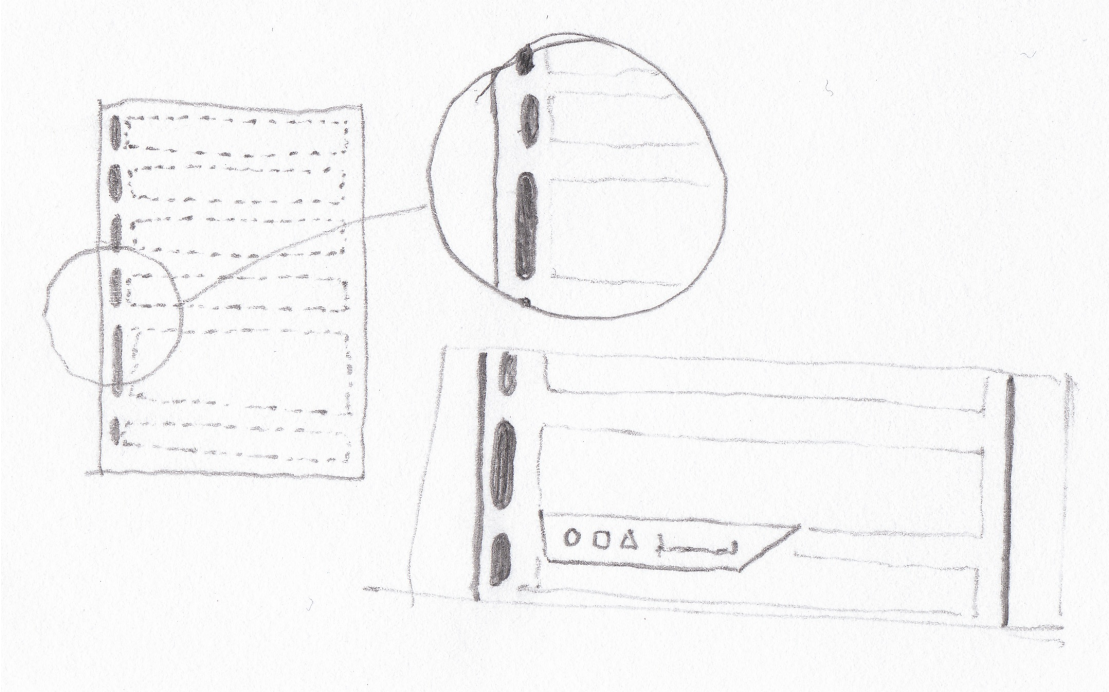


Glyphs will generally present interesting information and behaviour for an object. These will be in the form of buttons, graphs and text, anything that will help represent the underlying object. Objects will have other aspects to them, they will have properties and behaviours that can still be used in the user interface. For this, the handle contains a button that can display the details of the object, of course, this is displayed in another glyph.

These details will contain properties, which can be dragged off into new glyphs of their own, or methods, which can be invoked by preparing them. Preparing methods involves finding other glyphs or typing in arguments for the method. Once prepared, a method can be invoked which will usually create another glyph as the method's result.

Glyphs don't always have to stay in glyph space however, they can be placed inside other glyphs too. These are nested glyphs. A common example of this is when several glyphs can be placed into list of glyphs, which is a glyph itself. Each of these nested glyphs is displayed in the same way they would in normal glyph space, although, display options can be used so that only a reduced display is used so things don't get too cluttered. To further reduce cluttering, the handles of

glyphs aren't displayed when the mouse hovers over them, instead a tiny vertical bar on the left of each glyph can be clicked to show or hide the handle.



Scripts

The behaviour of glyphs, even the creation of glyphs can be automated using scripts. Actually, the glyphs themselves aren't automated, the underlying objects are automated and modified and glyphs just update their displays to represent the new changes. Scripts can also create new objects to be displayed (as glyphs) on screen. They come in two forms, either the script modifies existing objects in the system, or it just creates new objects based on the state of other objects. These two forms are called **commands** (to modify objects) and **formulas** (to just create new objects). Because formulas don't change anything in the system, they can be run dozens of times and still not change anything. Formulas then, can be rerun every time something in the system changes that would affect the result of the formula.

This is very similar to how a spreadsheet works: each cell can have either a value or a formula, and a formula can use the values or formulas in other cells to compute its own value. When a value changes in the spreadsheet, the formulas that depend on that value can be updated to reflect the new change.

Commands on the other hand are only executed once, when the user enters them. They can be run again if the user wishes of course, but this is not done automatically.

Scripts can use the values of other glyphs in the system by referring to them by name. This is the name that is displayed in the glyph's handle. The user can change this name so it makes more sense to his or her work.

Here's an example script (this one is a command), it creates a new list object and gives it a name "my-list".

my-list = list []

Here's another command that can be used to add a number to the list:

my-list.add (1)

And here's another command to view the number of items in the list:

my-list.count

Each of these scripts are displayed on the screen as glyphs of their own. The formula (my-list.count) is updated every time the list has an item added or removed from it. The two glyphs representing the commands can be re-evaluated as the user wishes: the first command to set the list back to containing nothing, the second to add new items again.

Scripts can be quite powerful, and the script language has dozens of features to make writing and maintaining scripts easy.

Layout

The glyphs discussed thus far can only be stacked vertically in glyph space. They can be contained within other glyphs however, giving some flexibility to the **container** glyphs in how they layout their nested glyphs. For example, a glyph may decide to layout its nested glyphs in a free floating 2 dimensional area, where glyphs can be moved around and over each other. Another scheme may allow glyphs to be included in a text document, flowing from left to right and down onto the next line.

The user can move around glyph space by clicking and dragging the right mouse button. By holding down the right mouse button and dragging the mouse right, the user can explore the area to the left. If the mouse is dragged down, the user can explore the area above.

The user can also zoom into glyphs on screen by double-clicking the right mouse button on a glyph. That glyph will now fill the entire screen, becoming the new glyph space, or it can represent an interface for a very specific task. The user can zoom out again by triple-clicking the right mouse button.

At the bottom (or the side) of the screen is a dock that can be used to place commonly used glyphs for ease of use. This dock is always stuck to the edge of the screen, following the user wherever he or she may zoom or navigate to. This dock may contain buttons (glyphs) that create new lists or tables; buttons to aid in zooming or navigation – for example, it may contain bookmarks to commonly used areas in glyph space.

Grammar

Most of this is founded on the idea that while software systems are regularly built using small functional components, the user rarely has a chance to make use of them. Instead, software is usually presented as large, complex and impenetrable systems that allow little flexibility or reuse. By breaking software down into smaller components that can be reused and recombined in new and useful ways, the user should feel more in control of their computer.

But it's not just about breaking software down into smaller components, it's also about having the tools to recombine those components, to build them back into larger software components as needed. This follows the idea that the most powerful user interfaces to date have (knowingly or unknowingly) used grammar as a way to combine different components into useful forms (the Unix shell has used this to great effect.) The smaller components may not be useful on their own but given tools to combine them into larger, more complicated components, the user can literally build software to meet his or her needs.

Grammars are usually recursive, in that the once combined, larger *composite* components share the same basic properties of smaller components. Thus, components together can form larger components, which in turn can form larger components again. While this may sound complicated to the average computer user, humans all possess the ability to understand grammar in natural language. The user should simply reapply this faculty to learning the grammar of the user interface.

Spatial Layout and Non-overlapping Visual Components

Two other ideas have had some influence in the design of the user interface; the first is that of overlapping windows, first seen in the 70s at Xerox with the intuition of Alan Kay. Overlapping windows, while helpful to advanced users, can confuse most by obscuring important information on screen. The worst offender is the modal dialog box, which not only visibly obscures information but also prevents the user from interacting with it. Overlapping windows, first developed to create more virtual area on the screen has made it more confusing.

The other idea is that humans have a very good sense of spatial awareness, which can help us to remember "where things are." This has been used in user interfaces representing the file system, allowing files to be placed in 2 dimensional space. The user remembers where the file was left last time, and can find it again without even having to remember its name.

We can increase the screen by increasing the height and width of logical workspace and arranging the visual components on the screen in a non-overlapping scheme. The user can zoom and or pan around this logical workspace in order to locate glyphs using their spatial awareness. The user navigates the wider glyph space as they would a map, making mental statements such as "the blog I was reading yesterday was up and to the left, next to where I keep my notes." Or, "I always store my photos next to where I leave all my image editing tools."

More abstract ways of navigation and finding things can be laid over this. A search bar can find glyphs by name; glyphs can be tagged or categorised; glyphs can be associated by their creation or modification date (this is likely to be a strong indicator that the glyphs were used in the same tasks); in a more advanced system, glyphs could be associated by the glyphs that created them – a kind of family tree for glyphs.

Of course, a different view and layout for each of these searches and associations would be helpful, again, they would be arranged and contained in their own glyphs. Once the glyph is found, double clicking it will zoom to the location of the

glyph where they last left it. It may make sense to put these searching and association tools in the dock for quick access.

Glyphs are self sufficient

A powerful notion embedded in objects and their glyphs is that they can describe themselves, present themselves and manipulate themselves. They are almost completely independent in that regard. They require almost no other user interface infrastructure to surround them.

This makes them somewhat portable, they can be dragged away from their original home and placed somewhere unfamiliar. They can be placed within other glyphs or alongside other glyphs and still retain full interactive and presentational functionality.