A Membrane with Parts: A new object model

Ted Kaehler

This material is based upon work supported in part by the National Science Foundation under Grant No. 0639876. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

VPRI Memo M-2009-014

Viewpoints Research Institute, 1209 Grand Central Avenue, Glendale, CA 91201 t: (818) 332-3001 f: (818) 244-9761

Memo: A Membrane with Parts: A new object model.

Ted Kaehler
November 2009

This memo was motivated by these questions:

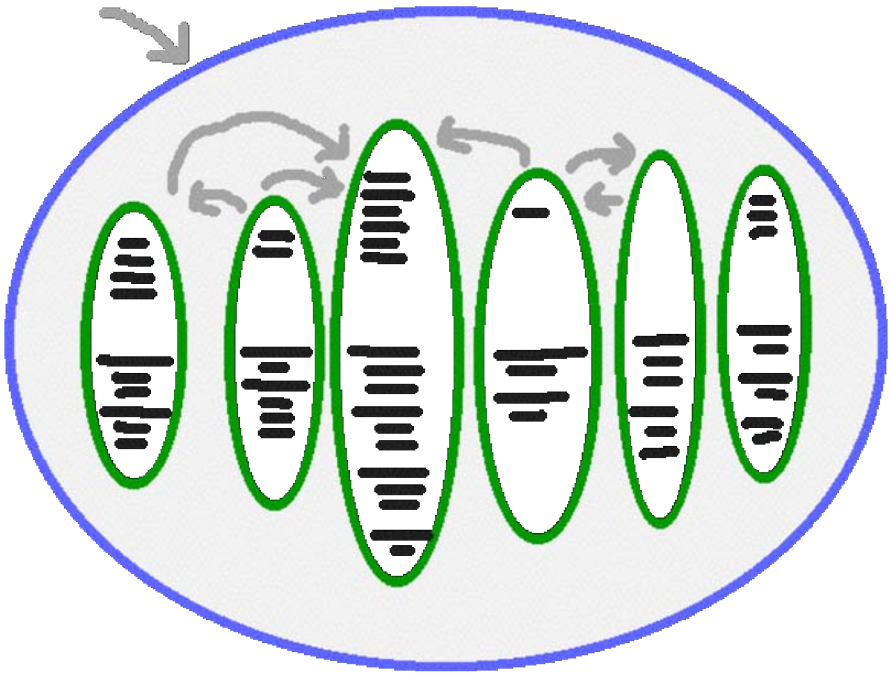
What is the next organizational level above a Smalltalk Object?
In a complex object with many subparts, it is hard to tell which parts belong to the main object and which are other independent entities.

How can we have object "Aspects" that really work?
Starting with Bobrow's PIE at PARC, there have been many implementations of objects with component parts. Many of these have been hard to use, and unclear.

What would "Stateful Traits" look like? Can we stop overloading inheritance?
We need to get away from using class hierarchy as the basis for specializing behavior. In Morphic, we add behavior to a graphical object by making a new subclass. In Squeak, there are 540 subclasses of Morph. We need a way to make these by composition of basic properties and behaviors.

How can objects be only as thick or as thin as needed?
"Thick Class Object" means that a basic graphical object comes with many capabilities built in. It is important to not carry along the parts needed for the full capability when an object does not use them. Objects fall many different places in the spectrum from thin to thick. They should only take storage for what they use.

Can we really have components that Mix and Match?
The Membrane solution comes is inspired by Yoshiki's iObject project. And, also by Ian's proposal that "Object = ID + Name Space".



A membrane object in blue, with component parts in green. Each component has instance variables (upper black lines) and methods (lower black lines). Communication between parts needs to be easy.

The Membrane is just an Identity Dictionary. The Membrane is an address space.
Parts inside are normal Smalltalk objects. Contents, border, fill, morphic role, layout, etc.
In code, "self" is the part, and "whole" is the outside membrane.

Each part uses the other parts heavily. References to the 'siblings' of a part is easy. That is we need an easy path to other parts from inside a part. In the part itself, getters for parts are generated automatically. Part X might say (self contents addLast: foo). Which uses this method in part X.

```
contents
    ^ whole at: #contents
```

Parts are as independent as possible. A given part can depend on a part of another kind being present. Many other parts depend on the "contents" part (that holds sub-objects).

Each kind of part has several versions. These are upward compatible. Borders may be simple or complex. Another part may depend on border being present, may install a default if it is not present, and will work when a more complex version is present.

Any part can be dropped in -- it carries all it needs with it, including state.

Motto: "Being a partner with other objects is more important than owning other objects."

Examples of parts:
In Etoys, each viewer category is a part.

Geometry	Script Control
Motion	Layout
Fill	Sound
Border	Logo
Color	Costume control
Pen	Overlap tests
Drag & Drop	

At my presentation, Alan suggested that communication between parts be by Linda-like publish and subscribe.

This memo is a clarification of my talk and slides from 21 Apr 2009.