What Smalltalk Rightfully Should Have Been

Michael T. Watters
mike@mwatters.net

March 7, 2014

Contents

1 Information Overload **2**
1.1 What's All This Got to Do With Smalltalk? 2
1.2 Encapsulation and Abstraction 2
1.2.1 Messages 2
1.3 Things with Attributes 2
1.3.1 Objects 2
1.4 Mental Models & Kinds of Things 3
1.4.1 Classes 3
1.5 A Word on Wooly Mammoths 3

2 What Smalltalk Provides **4**
2.1 Self-Contained Environments for Data and Code 4
2.2 That Sounds Familiar! 4

3 A Series of Unfortunate Events **8**
3.1 Could Smalltalk Have Been a Successful Operating System? 8
3.2 A Format for "Live Documents"? 8
3.3 A Language for the Web? 8

4 What Might Have Been **9**

1 Information Overload

From the moment we're born, we instinctively struggle to understand what's happening around us. We quickly develop skills to recognize and react to stimuli delivered from our sensory organs to our brains, an ongoing process which shapes the way we perceive and respond to the information in our environment.

1.1 What's All This Got to Do With Smalltalk?

Smalltalk is a computer programming language still in use today; it provides a useful system and method for organizing information. It was one of the earliest implementations of the Object Oriented Programming paradigm, and its concepts and features inspired a great many languages and programming environments which followed.

1.2 Encapsulation and Abstraction

Encapsulation and abstraction are the keys to usefully organizing information. As I write these words, I'm not thinking about the specific appendages of my body which are reacting to the neurotransmitter-driven electrical impulses originating in my brain, or the keyboard's electronic circuitry which translates the mechanical switch activations caused by my finger movements into electrical signals to be interpreted by my computer, or the operating system on my computer which receives those signals and delivers their assigned meanings to the programs with which I'm interacting, or how the CPU is updating locations in memory according to the various programs it's executing.

I'm thinking to myself: "I'm writing a paper using a computer", as that is a meaningful way of encapsulating my mental model of what I'm doing in a format which is convenient to remember and for communicating to others. The specific form of this activity is essentially a series of messages from my brain to my computer.

1.2.1 Messages

The quintessential feature of Smalltalk is the sending of `Message`s. A `Message` is a request for interaction between different parts of the system, and sending `Message`s is the only way for disparate components (`Objects`) to interact. This is a natural way to enhance modularity by enforcing separation between components.

1.3 Things with Attributes

An object in the real world is a thing with certain attributes and observed behaviors. We can change some of an object's attributes by doing things to it, and can learn about it by observing it and its interactions with other objects. We can categorize it and group it with other similar things.

1.3.1 Objects

In Smalltalk an `Object` is the unit of encapsulation for data and behavior. It may interact with other `Objects` by sending and receiving `Message`s. Data is private to an `Object`: the

only way to modify it is for the `Object` itself to do so in response to a `Message` from another `Object`.

1.4 Mental Models & Kinds of Things

When we think about a particular thing, our minds are operating on the mental models we've built to represent the various aspects of that kind of thing. We naturally group similar things together based on our observations and generalize abstractions which can be applied to group members, which saves time and mental energy.

When we see a tree, most of us won't consider or memorize every detail for future reference. We might make a note of any distinguishing features while thinking to ourselves "it's a tree", applying our pre-existing mental model corresponding to "tree-like things" to our perception of it.

These mental models allow us to make decisions when faced with incomplete information. Seeing a specific object which looks like a typical tree, we can quickly make useful assumptions about it, such as: it obeys the known laws of physics; it's alive; it's not dangerous; it can't move around or talk to us; and we can't eat it – because these things apply to all other objects we've classified as "trees" before.

1.4.1 Classes

To help manage the various behaviors and properties that `Object`s should have, Smalltalk defines a hierarchy of `Class`es. A `Class` defines the set of data which a member of the class may contain, and the set of behaviors (responses to `Message`s) which members are known to exhibit. All `Object`s are instances (members) of `Class`es, and their behavior is fully determined by their `Class`. Descendant `Class`es inherit data and behavior from their ancestors.

1.5 A Word on Wooly Mammoths

While not a perfect analogue, this hierarchical method of modeling data and behavior is similar to how we cope with information about things in the real world.

Consider a model involving prehistoric humans and wooly mammoths. Both are kinds of creatures which can wield pointy objects. Our limited understanding of living creatures indicates that one creature is hostile to another if it's wielding a pointy object and is not of the same species – potentially very helpful information when deciding whether to run away!

As expressed in the source code of Figure 1, our initial implementation of the model shows that a wooly mammoth is always hostile to a human (because it has tusks and a human is not a kind of wooly mammoth), so a human should always run away. A wooly mammoth should run away from a human if the human is carrying a sharp spear or other kind of pointy object.

While the model is useful in that it allows unrelated creatures to reason about each other using incomplete information, it fails to account for various details. For example, a human carrying a rope net in a hunting party might not want to run away, and a mammoth might not want to run away from a single human carrying a pointy object, or if it's with a pack of fellow mammoths.

As we learn more about a particular problem domain, we can update our class hierarchy appropriately to capture new information. For example, we could add an appropriate ancestor class to describe weapon-like objects, or we could add methods to the common ancestor of all physical objects and override them in objects describing physical objects sometimes useable as weapons. Smalltalk promotes this kind of incremental development.

2 What Smalltalk Provides

2.1 Self-Contained Environments for Data and Code

Using Smalltalk normally involves the use of a standard Smalltalk environment, which is a self-contained "image" containing a pre-defined class hierarchy along with whatever classes and data have been defined (or modified) and saved in the image by a user. An image behaves the same way on whatever host system is running it (typically by virtue of being interpreted by a virtual machine on each host platform). The entire image is by default open to user inspection and modification. Figures 2 and 3 show current (2014) and historical (early 1980s) screenshots of such environments.

2.2 That Sounds Familiar!

The "container environment for data and code" concept recurs throughout computing:

- My computer is a container for data and code which allows me to manipulate it;
- PDF documents contain typesetting instructions for text and other media, and may contain code to provide additional interactivity and functionality (via JavaScript);
- Microsoft Word documents contain data (text and formatting), and provide mechanisms for code to provide additional interactivity and functionality (via Visual Basic and COM add-ins);
- Modern web pages contain data (most often HTML) and code to provide additional interactivity and functionality (most often via JavaScript);
- Computer programs themselves are containers for code and data.

In all of these cases, a self-contained unit of information (code and data) can be transmitted from one host environment to another and reproduced. The receiving host environment contains a base level of functionality to interpret and display the data, and it can run the code which was included to provide additional functionality.

```
Object subclass: #Creature
    instanceVariableNames: 'heldObjects'.

hasPointyObject
    ^heldObjects contains: [ :x | x isKindOf: PointyObject ]

isHostileTo: anObject
    ^(anObject isKindOf: Creature)
        and: ((anObject isKindOf: self species)
                ifTrue: [ false ]
                ifFalse: [ self hasPointyObject ])

holdObjects: aCollection
    heldObjects addAll: aCollection

shouldRunAwayFrom: anObject
    ^(anObject isKindOf: Creature)
        and: (anObject isHostileTo: self)


Object subclass: #PointyObject
    instanceVariableNames: ''.

PointyObject subclass: #MammothTusk
    instanceVariableNames: ''.

PointyObject subclass: #SharpSpear
    instanceVariableNames: ''.


Creature subclass: #Human
    instanceVariableNames: ''.

Creature subclass: #WoolyMammoth
    instanceVariableNames: ''.

initialize
    super initialize.
    self holdObjects: (Bag with: MammothTusk new with: MammothTusk new)
```

Figure 1: A shorthand representation of Squeak Smalltalk code for the Wooly Mammoth model. The actual class and method definitions were created in a graphical editor inside the Squeak Smalltalk environment.

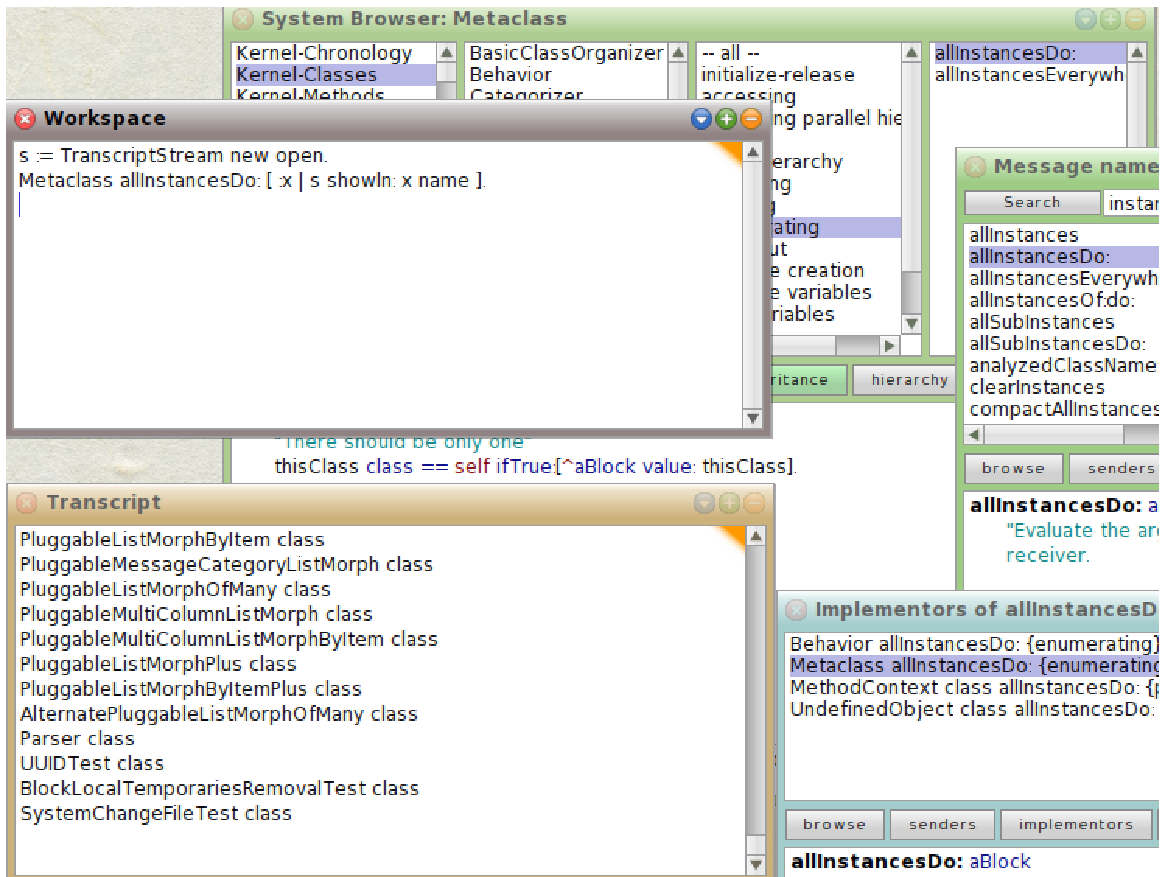

Figure 2: Examining the Squeak Smalltalk programming environment, a descendant of the original Smalltalk-80 environment. Everything is an object and all classes are open to inspection and modification. Image: MTW

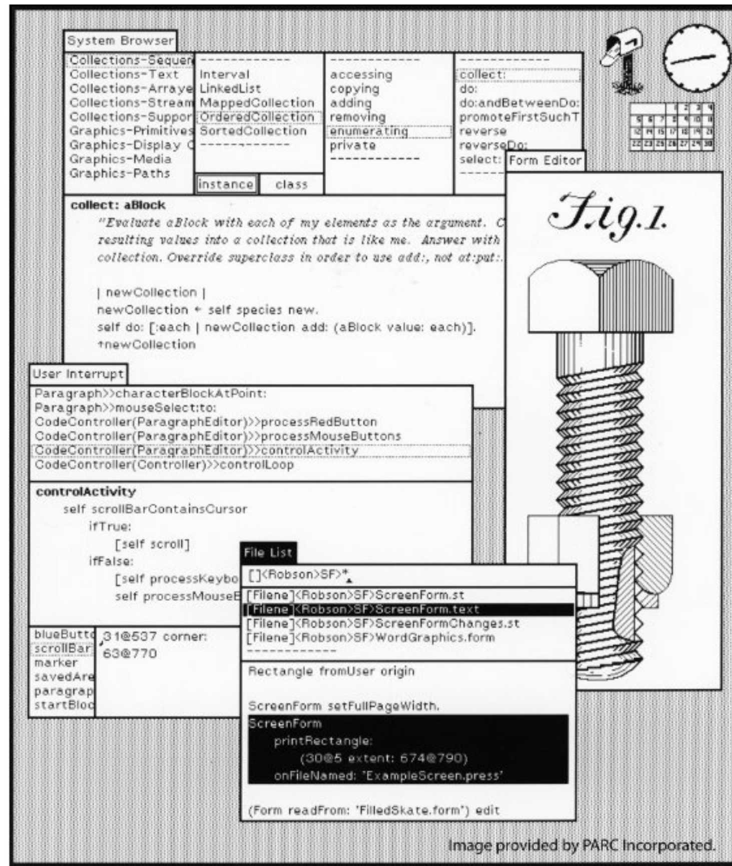

Figure 3: An original Smalltalk-80 environment running on a Xerox workstation in the early 1980s. Image: Computer History Museum

3 A Series of Unfortunate Events

3.1 Could Smalltalk Have Been a Successful Operating System?

According to Alan Kay, the original designer of Smalltalk, Steve Jobs tried to buy and/or otherwise obtain the Smalltalk technology from Xerox upon seeing a demonstration in 1979.[1] He was unable to do so. Had he succeeded, perhaps the original Apple Lisa and/or Macintosh would have been specialized machines for running Smalltalk environments instead of mere echoes of Xerox and Smalltalk concepts.

Still, computer operating systems come and go, and most users don't need (and shouldn't have) total control over how their system operates. An operating system based on Smalltalk and its philosophy would likely have been too "open" to be practical for most users (many of whom want to treat a computer merely as a tool for consumption which always behaves the same way, like a toaster or television).

3.2 A Format for "Live Documents"?

Apple began selling a Smalltalk-80 environment in 1984.[2] In 1987 it released HyperCard, a very popular "live document" environment with a number of similarities to Smalltalk (also based on partly self-contained "images" hosting data and code involving "messages to objects"), but without Smalltalk's extensive class hierarchy or simple syntax (HyperCard used interpreted English-like sentences in a bid to be more user-friendly).

If HyperCard had been developed as an extension to the base Smalltalk environment, perhaps entire generations of computer users would have come to appreciate Smalltalk's openness, power, and utility. Instead, HyperCard was killed off and largely forgotten, never to be faithfully recreated.

3.3 A Language for the Web?

Initial drawbacks of early Smalltalk environments included limited performance and relatively high memory usage.[3]

A startup company working on improving Smalltalk's performance had made some major progress by 1996, creating what may still be the fastest implementation of Smalltalk (known as Strongtalk).[4] Unfortunately, they were acqui-hired by Sun in 1997 and put to work improving Java instead.[5] One of the developers (Lars Bak) went on to lead the development of the V8 engine for JavaScript.

If the Strongtalk project had continued, a rich language (Smalltalk with optional type declarations) and environment which was speed-competitive with the dynamic languages of its era would have emerged and perhaps seen widespread use as a serious competitor to Java. What if "Smalltalklets" had been prevalent instead of Java applets?

---

[1] Google cache of http://www.smalltalk.org/smalltalk/TheEarlyHistoryOfSmalltalk_V.html on 2014-03-07.

[2] Google cache of http://www.smalltalk.org/smalltalk/history.html on 2014-03-07.

[3] http://en.wikipedia.org/wiki/Smalltalk on 2014-03-07.

[4] http://www.strongtalk.org/history.html on 2014-03-07.

[5] http://en.wikipedia.org/wiki/Strongtalk on 2014-03-07.

4 What Might Have Been

With its rich programming and data modeling environment (mapping naturally to the way we deal with information in the real world), and its (typically) image-based deployment mechanism providing identical behavior on multiple platforms, Smalltalk rightfully should have become the defacto standard for exchanging universal "documents with behavior".

Using the same concept of a class hierarchy, there could be multiple "well-known base environments" which inherit behavior from ancestors (all the way to an appropriate root environment, such as an ANSI standard). There could be user-extensible standard environments for things like word processing documents, spreadsheets, and hypermedia (web pages); each of these would be defined by their differences from a suitable ancestor environment. Similar kinds of documents could share common behavior, greatly enhancing options for code reuse and interactivity among different kinds of data.

Under this model, most documents could be packaged as "Smalltalk image deltas", containing only a reference to a well-known environment and the necessary code changes and data specific to the document.

The concept of "file types" is today an approximation of this kind of universal system: a file's type is a reference to a "well-known environment" (a program implementing the format), but each file type has its own method (possibly none) of representing "additional behavior" to be made available to the user accessing the document represented by the file.

A free, widely available, and performant implementation of Smalltalk with a well thought-out class hierarchy would have provided a compelling foundation upon which such a "universal document" system might have been built. But it never came to be.