Avocado: Programming JavaScript in a Self-ish Environment

Adam Spitz (adam.spitz@gmail.com) and Josh Flowers

Abstract

Avocado is a programming environment for JavaScript, written in JavaScript. In Avocado, all programming is done live while the program is running, and the environment tries to immerse the programmer in a world where his JavaScript objects feel like tangible things. The JavaScript language is missing several facilities that would greatly enhance its ability to support this style of programming environment, but nevertheless we've found it to be a productive and fun way to create JavaScript programs.

Introduction

We're big fans of Smalltalk-style "live" environments, with all coding done while the program is running (as opposed to "text-file-based" environments where the programmer has to stop the program and change the text files and restart the program). We feel this way partially because the more-immediate feedback makes us more productive, and partially because the human brain is already accustomed to living in a real world that's always running - it feels more *natural* to work in a live environment. (If you want to change the color of a wall in your house, you don't tear down the house, modify the source code, and then re-build the house.)

In particular, we're fans of direct-manipulation programming environments like Self, where programming feels more like "getting your hands on your objects" than like "getting your hands on tools that are somehow connected to your objects which live somewhere deep inside the bowels of your computer."

The JavaScript language is a lot like Self in many ways. It's object-oriented, it's dynamically-typed, it's even prototype-based. We don't like the JavaScript language as much as we like the Self language, but... well, JavaScript is already built into every web browser everywhere. Like it or not (and we do, in fact, like it a lot more than we expected to), JavaScript is the client-side language of the web.

So we wondered: could JavaScript be programmed using a live environment, in the tradition of Smalltalk and Self?

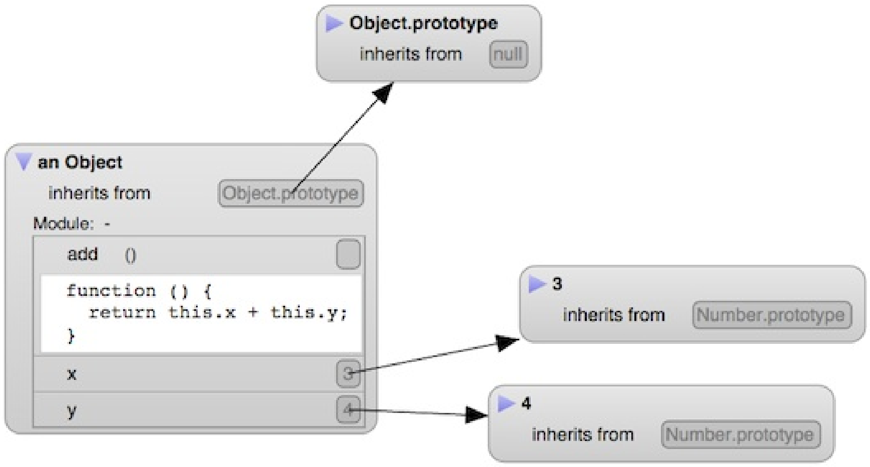
Avocado

Avocado (http://avocadojs.com) is a programming environment that we built in JavaScript, for JavaScript. (Avocado is open-source, under the MIT license; the website contains a link to Avocado's GitHub repository.)

All programming in Avocado is done live, at runtime. You point your web browser at the Avocado URL, and the programming environment appears inside the browser. Inside the programming environment are things like objects and functions. An object looks like a box. A pointer looks like an arrow from one box to another. You never see anything that looks like a big text file containing all your code.

Direct manipulation of objects

Avocado tries to avoid presenting itself as a set of "tools." Rather, Avocado tries to immerse you in a world where you feel like you're interacting directly with your JavaScript objects. There is no "code browser" tool - just the objects themselves, and you can pick them up and move them around (just as you're accustomed to doing with objects in the real world). As in Self, programming is done by getting your hands on your objects and doing things to them. If you want to change the object (say, by adding a property), you go right up to the object and tell it to add a new property. If you want to ask the object a question (i.e. call one of its functions), you go to the object and type in your question. If you ask the object how it fits into the inheritance hierarchy, the objects whoosh around on the screen to arrange themselves into a tree shape.



```
function () {
  return this.x + this.y;
}
```

Your task, as a programmer, is to construct and manipulate the objects of your program. If the program you're writing is a zoo simulator, your job as a programmer isn't about the Code Editor and the Logging Console and the Class Browser and the Inheritance Tree, it's about the Monkeys and Cages and Pathways and Elephants. So *those* are the objects that Avocado tries to present as being real and tangible.

Transporting objects to other worlds (via .js files)

Of course, after you've built your live JavaScript objects inside your web browser, you need a way to transport those objects into other people's browsers. That's what text files are good for. Avocado has a "transporter" system that you can use to save your program in .js files, so that they can be written to disk and later loaded into other web browsers.

Avocado's transporter system is based on Self's; for more information on how it works, see David Ungar's 1995 paper, Annotating Objects for Transport to Other Worlds. One of the interesting ideas behind the Self transporter is that a web of live objects does not quite have enough information to capture the programmer's intentions; the objects need "annotations" (meta-data) containing information like:

- Which "module" (i.e. which .js file) should this property be saved to? (This allows the programmer to split up the program into multiple files, each of which can be loaded separately if desired.)
- Should the property's current value be saved, or should the property be initialized to some "initialization expression"?

Avocado allows the programmer to specify this information, and stores it on each object, in a property called "__annotation__". (This property is non-enumerable, so it's invisible to the programmer unless explicitly asked for.)

User interface

Avocado's user interface is based on SVG (Scalable Vector Graphics), which is a web standard and hence works identically in every browser except when it doesn't. Avocado seems to run well in Safari and Chrome and Firefox.

Avocado's UI owes a huge debt to Dan Ingalls' Lively Kernel system - Avocado started out as a fork of the Lively Kernel code base.

Like Self, only less so

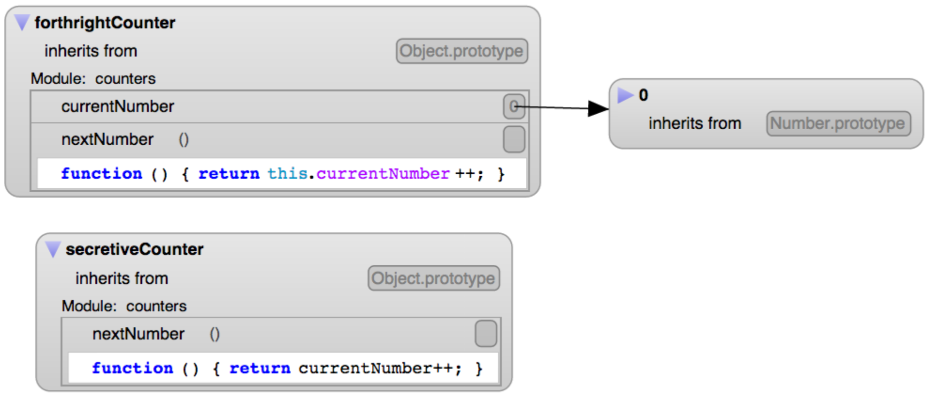
Here are some of the obstacles we've encountered in trying to use JavaScript in a live way.

No reflective access to local variables

The JavaScript language has reflective facilities for accessing the properties of an object. The "Object.keys" function is used by the Avocado environment to obtain an array of the names of the object's properties, and the "obj[key]" syntax can be used to access the values of those properties. Unfortunately for Avocado, JavaScript provides no reflective way to access the values of variables in a function. For example, here are two ways of creating a "counter" object:

```
var forthrightCounter = {
  currentNumber: 0,
  nextNumber: function() { return this.currentNumber++; }
};

var secretiveCounter = (function() {
  var currentNumber = 0;
  return {
    nextNumber: function() { return currentNumber++; }
  };
})();
```



```
function () { return this.currentNumber ++; }
```

```
function () { return currentNumber++; }
```

Both forthrightCounter and secretiveCounter have a "nextNumber" method that will return the sequence 0, 1, 2, 3... when called repeatedly. But only forthrightCounter can be examined at runtime to determine the value of its currentNumber variable.

JavaScript programmers often use the second style as a way to do information-hiding. Unfortunately, this prevents live programming environments like Avocado from being able to show you a complete picture of your objects. So Avocado can only support the first style of programming.

No multithreading

Avocado has no debugger (we rely on the web browser's built-in debugger), for two reasons:

- Any debugger written in JavaScript would be unable (for the reasons mentioned above) to access the values of local variables.
- JavaScript's multithreading facilities (i.e. "web workers") are very limited: web workers cannot access the DOM, and are also limited in their ability to access other objects.

When a piece of JavaScript code encounters an error, we can try to use the web browser's built-in debugger to find the problem, but we can't interact with the running Avocado world as we're doing so (since the sole JavaScript process has been frozen for debugging), and the web browser's debugger doesn't have Avocado's facilities for visualizing and manipulating objects.

No snapshotting

In Smalltalk or Self programming, on top of the "transporter"-style saving mechanisms discussed above (which involve saving a subset of the objects in the heap in a format that can be loaded cleanly into other heaps), it is also possible to save a "snapshot" of the entire object heap so that it can be reloaded as a whole. This facility is very useful when doing live programming, because there are often transient objects that aren't part of any module (and hence won't be saved by the transporter); a snapshotting mechanism allows you to save your place (including all these transient objects) and return to exactly where you left off.

Avocado does not have such a snapshotting facility, because, as far as we know, no web browser offers the ability to take a reloadable snapshot of the entire object heap. So we make do with the transporter.

Conclusion

We've actually been very impressed with JavaScript's ability to support a live style of programming, despite feeling like we're trying to smash a square peg into a round hole. JavaScript is missing several capabilities that would allow it to truly support live programming - reflective access to local variables, unrestricted multithreading, and whole-heap snapshotting - but we've nevertheless managed to create a live programming environment that we've found usable and fun. It's a pleasure to be able to work on JavaScript code while it's running, using a visual environment that lets us feel like we're immersed in a real world full of JavaScript objects.