What Objective-C 3.0 Could Have Been

Adrian Kosmaczewski

2022-12-09

There's a parallel universe, with a parallel WWDC 2014, in which instead of Swift, developers got Objective-C 3.0, and this is what it would have looked like. It's the same parallel universe where Russia doesn't annex Crimea, by the way.

To begin with, Objective-C would have made header files entirely optional, and if you didn't provide one, they'd be automatically generated from the implementation file. Just write your `*.m` files and save.

The `id` type is great but let's be honest, it does not help much. It's a big part of what I called the "west coast" compiler mentality a few years ago[1]. Instead, Objective-C 3.0 introduced the new `auto` type indicator, which triggers type inference, to make the compiler verify that the methods you're calling actually exist; and the type of the `auto` receiving variable would be whatever the method returns.

Objective-C 3.0 would have dropped class prefixes for actual namespace support. And since we're at it, it would have introduced custom operators. Developers love syntax sugar.

A package manager. Yes, Objective-C 3.0 would have brought that, of course. Hosted by Apple, available to developers registered in the App Store program as an added value. With security checks and everything, thank you so much.

You know what? Let's be crazy. Objective-C 3.0 would have removed the requirement for **square brackets**. Yes, I've said it. Extending the dot syntax for method calling. And since we're at it, drop the `@` sign for strings, arrays, and dictionaries. Seriously, just drop it. Very few people uses C-strings in iOS apps, seriously, and if you *really* wanted to use them, surround the code with one of the newly introduced `c {…}` blocks, and that's it. Whatever you put inside is pure C. Oh, and you've got another block called `cpp {…}` for… you guessed it, C++ code. ABI stability FTW.

In the same vein, let's not forget about the new `rust {…}` block that provides interoperability with this new little language from Mozilla. Who knows, it could be even used in an operating system kernel in the future; Rust looks really promising.

Since we have a `c {…}` block for pure C, how about we drop the `*` in front of variables? We know that Objective-C objects live on the heap (except for blocks)

[1] /blog/the-developer-guide-to-migrate-across-galaxies/#4-objective-c-aka-coolia

so just drop the asterisk. Now Objective-C 3.0 really looks like Objective-J[2], well, apart from the square brackets, which are gone.

What else? Inspired by Go or D[3], Objective-C 3.0 would have included built-in unit tests, using the new… `unittest {…}` block. To stop people from arguing about source code formatting, the new `objcfmt` tool would automatically format code, just like Go or Rust would do. KVC would have evolved into something akin to what LINQ provides for C#, providing live queries with compile-time checks, for more complex data manipulation at runtime. And KVO would have evolved from the Observer pattern to a React-enabling technology, built around data streams.

And because all languages end up in a Kubernetes cluster these days, and because Objective-C 3.0 is a language based on messaging[4], it would naturally compile source code into microservices. Yes, that's right; it would compile your application code into several separate containers[5] ready to run on your orchestrator of choice, and to help developers, it would do the same but into a single binary for local debugging. Of course, production builds would be `scratch` or `alpine` containers, small and with as few dependencies as possible.

All of these new Objective-C 3.0 features would have been enabled via *ad hoc* special compiler arguments; that is, Objective-C 3.0 would have been 100% compatible with Objective-C 2.0 code, with new features enabled by default on new projects. You could start adopting them little by little, one at a time, without breaking your code.

In any case, this parallel Objective-C 3.0 would have kept the dynamic runtime, adding a little bit of static type checking for those who would really want it and need it. Boring[6], but great.

Keeping Cocoa, evolving it, and building on top of it; not just throwing it away.

But, of course; saying any of this (even though I'm not the first[7]) is simply akin to heresy[8]:

> Have you ever felt developers might get a little stigmatized for using Objective-C, or talking about it on social media?
>
> Steve Troughton-Smith: A little? It's deeply unpopular to use ObjC or say you like ObjC over Swift. Swift has a truly massive hype train, and you definitely don't want to stand in the way of it.

I *never* really enjoyed writing Swift code. I can't help but to see in it nothing else than the arrogance[9] of a company that used to beg for developers to pay attention to it.

[2] https://www.cappuccino.dev/learn/objective-j.html
[3] /blog/d-or-what-go-may-have-been/
[4] /blog/microservices-or-not-your-team-has-already-decided/
[5] /blog/reusing-apps-between-teams-and-environments-through-containers/
[6] /tags/boring-languages/
[7] https://mjtsai.com/blog/2014/10/14/hypothetical-objective-c-3-0/
[8] https://www.hackingwithswift.com/articles/27/why-many-developers-still-prefer-objective-c-to-swift
[9] /blog/courage/

In 2009, the iPhone simulator launched in milliseconds while[10] the Android emulator took ages to do so. In 2017 it was exactly the opposite[11]. Funny how the tables have turned.

Anyway[12], what do I care.

[10] /blog/that-mobile-programming-mess/

[11] /books/Android_for_iOS_Devs/Android_for_iOS_Devs_Kotlin_Ed_2018.html#chapter_toolchain

[12] /blog/migrating-from-macos-to-linux/