Dashboard vs. Konfabulator

by John Gruber

A sliding puzzle. A calculator. A clock. A little notepad. Tiny little applets—little pieces of software that are something less than full applications themselves, but which run alongside real apps and are easily accessed at any time.

Obviously, Apple ripped off the idea for Dashboard. Stolen wholesale, without even the decency to mention where they took the original idea.

Which, of course, would be the desk accessories from the original 1984 Macintosh—conceived by Bud Tribble and engineered (mostly) by Andy Hertzfeld. Here's Hertzfeld on the origin of desk accessories:

> Bud Tribble was usually on an even keel, but one afternoon in the fall of 1981 he came into my office, unusually excited. "You know, I've been thinking about it. Even if we can only run one major application at a time, there's no reason that we can't also have some little miniature applications running in their own windows at the same time."
>
> That sounded intriguing to me. "What kind of little programs? How are they different?", I wondered.
>
> Bud smiled. "You'd want tiny apps that were good at a specific, limited function that complements the main application. Like a little calculator, for example, that looked like a real calculator. Or maybe an alarm clock, or a notepad for jotting down text. Since the entire screen is supposed to be a metaphorical desktop, the little programs are desk ornaments, adorning the desktop with useful features."

The post-WWDC peanut gallery is atwitter with the idea that Tiger's Dashboard is a blatant rip-off of Konfabulator. You can't read *anything* about Dashboard without hearing that it's a Konfabulator rip-off.

Bullshit. *Dashboard is not a rip-off of Konfabulator.* Yes, they are doing very much the same thing. But what it is that they're doing was not an original idea to Konfabulator. The scope of a "widget" is very much the modern-day equivalent of a desk accessory.

What is original to Konfabulator? That its widgets are based on a scripting language? That's not original. There are a slew of other scripting runtimes that allow you to create little-applets-in-windows. E.g. on the old Mac OS, there was OneClick.

According to the Konfabulator web site:

> What sets Konfabulator apart from other scripting applications is that it takes full advantage of Apple's Quartz rendering. This allows Widgets to blend fluidly into your desktop without the constraints of traditional window borders. Toss in some sliding and fading, and these little guys are right at home in Mac OS X.

That sounds right—the most striking thing about Konfabulator is that its default widgets look cool. Anti-aliased type. Round corners. Transparency. In short, they're clearly the work of a talented artist, that being Arlo Rose, who, before co-creating Konfabulator, had in fact worked as a human interface designer for Apple in the 90s, and then went on to co-create the Kaleidoscope theming hack for the old Mac OS.

The crux is that what makes Konfabulator stand out is that its widgets look "right at home in Mac OS X". But is it any wonder that Apple's Dashboard gadgets look "right at home" on Mac OS X, too?

Nomenclatural note
Konfabulator calls its applets "widgets". Apple's Dashboard developer documentation refers to its applets as "gadgets". However, at the WWDC keynote and at the Tiger Preview section of Apple.com, Apple refers to Dashboard's applets as "widgets". Until this is clarified, I'll use "widgets" to refer to Konfabulator's dinguses; and "gadgets" for Dashboard's.

It's more than a little reminiscent of the supposed "controversy" over Panther's greatly-improved Command-Tab switcher, wherein the fact that it (the Panther switcher) visually resembled Proteron's Liteswitch X switching utility caused a bunch of knickers to get knotted. Most infamously, Proteron developer Sam Caughron posted an "open memo" wherein he took the position of victim—that he was an innovator done wrong by Apple.

But the functionality of Liteswitch X—using keyboard shortcuts to cycle through running apps—is over 20 years old, and present in every major desktop OS for the last decade. The argument that Apple had "ripped off" Proteron more or less hinged in the idea that the visual resemblence between the new built-in Panther switcher and Liteswitch X couldn't be a coincidence. Of course it wasn't coincidence—but that doesn't mean it was a "rip-off". I wrote in October:

> Part of Caughron's argument is that Panther's implementation is "a near pixel duplication of [. . .] LiteSwitch X"—meaning that they look the same. Proteron did a fine job designing how LiteSwitch X looks, but it is not a novel design—it looks good because it looks like something designed by Apple. Is it any surprise that Apple's new switcher does too?

Also similar to the command-tab/Liteswitch X hubbub is that the consternation isn't coming from the developer community, but rather from the peanut gallery of non-developers. Mac developers are not up-in-arms about Dashboard; in fact, the 9 a.m. Dashboard session Tuesday morning at WWDC filled up so quickly that they had to turn people away, long before the session started. (Part of this is excitement over Dashboard itself; part is that there's a contest for the best gadgets developed this week at WWDC, with the prizes being a 15" PowerBook and a 40 GB iPod.)

Look Under the Hood

The idea, repeated ad nauseum in the initial reaction to Dashboard, is that Apple should have done the "right thing", where "right thing" means "bought the rights to Konfabulator instead of implementing Dashboard on their own".

But this implies that Dashboard is an under-the-hood equivalent to Konfabulator. They both use JavaScript, right?

Think again. Dashboard, from a how-it-works perspective, is very much different than Konfabulator.

Konfabulator is a self-contained JavaScript runtime engine, using its own JavaScript interpretter. (I suspect it's based on one of the various open source JavaScript implementations, but I see no credit or reference to such on the Konfabulator web site or in their documentation. If anyone knows, please let me know.) Konfabulator UI layouts are specified in a custom XML format. I.e.:

Konfabulator = (Custom XML format) + (Custom JavaScript engine)

Dashboard, on the other hand, is based on WebCore, the underlying open source layout and scripting engine behind Safari. Dashboard gadgets are indeed scripted using JavaScript, the same language used by Konfabulator, but Dashboard uses the JavaScript engine that's built into the system. And for UI layout, Dashboard gadgets are specified using HTML and CSS—using the same rendering engine as Safari.

Dave Hyatt writes:

> I wanted to blog briefly to clear up what the widgets actually are written in. They are Web pages, plain and simple (with extra features thrown in for added measure). Apple's own web site says "build your own widgets using the JavaScript language", but that's sort of misleading. The widgets are HTML+CSS+JS. They are not some JS-only thing.
>
> In other words, each widget is just a web page, and so you have the full power of WebKit behind each one... CSS2, DOM2, JS, HTML, XMLHttpRequest, Flash, QuickTime, Java, etc. I'll have a lot more to say later on, but I thought it important to clear that up right up front, since a lot of people were asking me about it in email and such.

Do you see how huge this is? How it opens the door to gadget development to anyone with web design experience? Indeed, I've read the preliminary Dashboard developer documentation (generously provided by a source attending WWDC), and it is outstanding from the perspective of making gadgets easy-to-create.

The idea that Dashboard is derivative because it's scripted via JavaScript is missing the point. Dashboard isn't using JavaScript just to use JavaScript—it's using JavaScript because Dashboard gadgets are little floating Web Kit views.

Plus, Dashboard gadgets are further extensible using Cocoa. You don't *need* to use Cocoa—fully functional gadgets can be made using nothing more than HTML, CSS, and JavaScript—but the option to use Cocoa is there for doing things JavaScript alone can't do.

Thus, Dashboard is clearly an extension of Mac OS X system-level technologies: Web Kit for layout and scripting; Exposé for the Dashboard window layer; and Cocoa for advanced functionality. Dashboard is the result of advanced Mac OS X technology in action.

Konfabulator, on the other hand, was designed from the start with platform portability in mind. (A port to Windows was announced back in December.)

That's not to say Konfabulator widgets can't use Mac-specific features; e.g. the ability to call AppleScripts. But in terms of the underlying runtime engine, none of it is based on Mac OS X-specific technologies.

Yes, no argument about it, Dashboard is a Konfabulator-killer. Their gadgets and widgets are serving the same exact purpose. I.e., the answer to the question *What do they do?* is the same. But the answer to the question *How are they doing it?* is completely different.

To assume that Apple started with the idea of putting Konfabulator out of business for spite is to ignore the more obvious reason why Apple didn't buy out Konfabulator: they had something completely different in mind engineering-wise.

Do you see how it's entirely plausible that Apple's decision to base Dashboard on Web Kit and Cocoa was based on purely technical reasons? Konfabulator isn't a *product*—it's a *platform*. Konfabulator itself does nothing other than provide an environment and API for widgets.

Adding a new platform layer to the system is a serious decision and commitment. If you're still willing to argue that Apple should have bought Konfabulator as the basis for Dashboard, you're implicitly arguing that Apple should be more concerned about being nice to third-party developers than they are about the quality of the engineering undergirding their platform.

Konfabulator is not a lightweight or small-footprint environment—every Konfabulator widgets runs as a separate process, with its own runtime environment in memory. Most Konfabulator widgets use more memory than typical full-blown Mac OS X applications. Not just Konfabulator as a whole—but *each widget*. Install it, fire up Process

Viewer, and see for yourself. (Ironically, the Konfabulator "CPU Portal" widget seems to leak memory.)

For all of the armchair critics who claim to know that Apple "should have" bought Konfabulator to serve as the basis for Dashboard, I ask:

- Have you *used* Konfabulator? If so, have you measured its memory consumption?
- Do you think Apple's OS engineers should be concerned about performance and resource consumption?
- Do you think you know more about performance and resource allocation than Apple's engineers?
- Do you believe reasonable engineering opinions can be drawn by looking at screenshots?
- Do you realize how insulting it is to have someone who knows *nothing* about the details of your work tell you that they know better than you do?

Web Developers as Gadget Developers

It's not hard to speculate on the reasons why Dashboard is based on WebCore technology. First, it's already there—the JavaScript engine and HTML/CSS renderer are built-in components in Mac OS X. Second, basing Dashboard development on industry standard web technologies means that anyone who knows how to design a web page knows how to design a Dashboard gadget.

In terms of attracting hobbyist-level developers, Dashboard is going to be an order of magnitude more approachable than Cocoa application development. If you know HTML and CSS, you can design a gadget layout. If you know JavaScript, you can program a gadget.

I can't think of any possible way that gadget development could be based on something that would enable more Mac nerds to get up and running writing their own gadgets. That the presentation is based on the *same* HTML/CSS rendering engine as used by Safari makes it possible to test your layouts by loading them in the browser. (In fact, that's exactly what the Dashboard developer tutorial recommends.)

This, in fact, is the single biggest difference between Dashboard gadgets and the aforementioned desk accessories in the original Mac OS. Because of the tight memory constraints of the original Mac (only 128 *KB* of RAM, and 400 *KB* of disk space per floppy), desk accessories needed to be as small and lightweight as possible. Which meant that they were mostly written in 68000 assembler code.

Andy Hertzfeld (him again) used Pascal to write the first version of the Puzzle desk accessory in 1982, two years before the Mac shipped. But at 6 *KB*, it was deemed too big:

> Jerome Coonen, the software manager, came by my cubicle one morning to tell me that they decided not to ship the puzzle, partially because of the game-like perception, but mostly because it was just too big. Applications were very tight on RAM, and the puzzle was one of the biggest desk accessories because it was written in Pascal. At over 6K bytes, it also ate into the available disk space.
>
> I liked the puzzle and I didn't want to capitulate to the buttoned-down, all business view of the customer, so I told Jerome, "You know, the puzzle doesn't have to be so big. I bet I could rewrite it and get it to take up less than 1K bytes. Would you keep it if I got it that small, or is it really the other issue?"

> Jerome thought about it, and then told me if I could get it down to 600 bytes or so, it would be in the release. The only problem was I had to get it done over the weekend, because they had to send the manuals out to the printer soon, and there was plenty of other stuff for me to work on.
>
> Of course, I couldn't resist a challenge like that. It only took a few hours on Saturday to recode it in assembly language, and get it down to the required 600 bytes, since it no longer had to link with the bulky Pascal code. I proudly showed it off to everyone on Monday, and it did make it into the first release, and stayed there for many years, although it was completely rewritten a couple of times, for various reasons.

Thus, the difference between the old world of desk accessories and the upcoming world of Dashboard gadgets: there are only a handful of people in the world who could write a puzzle game in the form of a Macintosh device driver using just 600 bytes of 68000 assembler; there are hundreds of thousands who can write one using HTML/CSS/JavaScript.

Desk accessories were harder to write than real Mac apps; Dashboard gadgets are far *easier*, and are based on web standards that people already know. Dashboard gadgeteering is going to be hugely popular—largely because it's based on web development technologies people already know.

Cry Me a River

In an interview with CNet reporter Ina Fried ("Developer Calls Apple's Tiger a Copycat"), Arlo Rose said:

> "It's insulting, is what it is," Rose said in a telephone interview. "They could have at least offered to work with us or to buy it."

It's natural for Rose and co-creator Perry Clarke to be upset; Dashboard is clearly a Konfabulator-killer. (Rose to Fried: "We're either going to have to move to another platform or work on some other project.")

But what's the argument that Apple has done something wrong? That if a third-party developer something first, Apple should never step on that developer's toes? Ever? No matter if Apple is already working on something similar but better? No matter if Apple can provide a *significantly* better implementation? No matter if it's something that would be a natural fit as an official component of the OS?

Scott Knaster—long-time Mac programming author—echoes Arlo Rose's opinion of "the least" Apple should have done:

> Dashboard is interesting. It looks like a complete ripoff of Konfabulator, right down to the word "widgets". Shame, Apple. At least buy the guy's technology.

My question: Even if Apple considered Konfabulator's technology undesirable?

In the lean years between John Sculley's exit and Jobs's return, Apple indeed seemed to follow this dictum of not stepping on Mac developers' toes. And you know what we got? Years of stagnation with System 7.5.

I am not advocating that Apple mindlessly trample on independent Mac developers—what I'm saying is that Apple cannot be afraid to improve the Mac OS as best it can, even if that means going places where small indie developers have gone first.

In cases where indie developers have created products that meet Apple's own high standards, the company has shown that it's willing to embrace existing solutions.

Most famously in recent years, iTunes—which started as SoundJam. Apple bought the rights, hired the development team (lead developer Jeffrey Robbin is now a serious big shot at Apple, helping lead all of Apple's various music-related initiatives).

Most *infamously*, the Watson/Sherlock controversy. Except note that Apple offered Watson developer Dan Wood an engineering position on the Sherlock team, which Wood declined. This is of course contrary to the popular misconception that Apple "blindsided" Wood with Sherlock 3 (which had been in development before Watson debuted). Wood wanted compensation for the existing work he'd done on Watson, not just a job that would pay him for future work on Sherlock. Twice offering Wood a job on the Sherlock team doesn't qualify as oppression.

That Konfabulator made it easy to create custom-designed widgets—does that mean Apple was obligated never to ship something that made it easier? With a smaller memory footprint? Isn't it plausible that this was part of the plan for WebCore and Exposé all along—and that obtaining the rights to Konfabulator wouldn't have helped them with this plan at all? Please keep in mind that just because they do the same thing from a user's perspective doesn't mean they do the same thing at all in terms of under-the-hood implementation.

Apple may hire you to work for them. They may buy your code if they want it. But they will not pay you off just for an *idea*, nor should they be expected to.

There aren't that many brand-spanking new ideas under the sun, and Konfabulator isn't one.

The idea of "little apps" began with Apple's desk accessories, 23 years ago.

The idea of a scriptable runtime for displaying little windoid applets is not original to Konfabulator. (C.f. DesktopX, a similar product for Windows that shipped in 1999.)

Even Konfabulator's obsessive focus on the pixel-perfect cosmetic appeal of its widgets is not novel. The original Mac desk accessories were designed pixel-by-pixel. The standards for GUI showboating have certainly changed since 1984, but that's only as a function of hardware capabilities.

Susan Kare's amzing layout for the original Control Panel desk accessory involved no text labels, only graphics (screenshot from Folklore.org):

And, according to Hertzfeld (who else?), Steve Jobs personally designed every pixel-level detail of the original Calculator.

At the end of my article on the Liteswitch X "controversy", I wrote:

> . . . when a utility is designed to compensate for a hole in Mac OS X, the developer should not expect the hole to remain unfilled by Apple forever.

If you're going to compete, you've got to accept that you might lose. Konfabulator is a software platform; Rose and Perry decided upon a rather closed platform model: the only way even to run a Konfabulator widget is to purchase a license to the runtime.

In my review of Konfabulator 1.0, I wrote:

> What's most perplexing to me is what Konfabulator does not include: an IDE. There is no visual widget editor or source code editor. They do include fairly comprehensive documentation on the widget XML structures and the supported JavaScript syntax, but when it comes to creating your own widgets, you're on your own. Arlo Rose, writing in the Konfabulator support forum, wrote:
>
> > Photoshop and BBEdit are your friends. :-)
> >
> > We are working on a visual editor, but we are net [*sic*] yet sure when it will be released.

> So they're obviously aware of the need for an IDE. But I can't help but think it would have better had they waited for the IDE before they started selling Konfabulator. As it stands now, you need Konfabulator just to run a widget. Let's say I have an idea for a widget-type application that does something very cool. I wouldn't write it for Konfabulator, because I wouldn't want to limit the audience to those who've paid $25 for Konfabulator.
>
> It makes more sense to me to give away the runtime engine, and let everyone run widgets for free. Then charge money for the IDE (and charge a bit more than $25). This makes widget-writing more appealing—you can share your work with the whole wide Mac world, not just the pixel-tweaking Konfabulator Klub.

A widget construction IDE would have made Konfabulator a product, not just a platform for pixel-pimping GUI tweakers. If they'd gone that route, they'd still have a future in the post-Dashboard world—by expanding the IDE to produce Dashboard gadgets in addition to Konfabulator widgets. (Note to Mac developers looking for software ideas—hint, hint).

It was their choice to compete only on the basis of providing a widget platform.