Creating BridgeTalk

Rob Corell

The Problem

My team pulled the Photoshop file browser out of Photoshop and made it into its own fully-fledged application, Adobe Bridge, intended to replace the Finder/Explorer for Creative Suite users. But it couldn't just be a siloed, standalone app; it needed to integrate seamlessly with not just Photoshop but the rest of the Suite. We needed a way for Bridge to communicate with other Adobe applications and control them to some lesser or greater extent. Put another way: we needed to select an IPC (Inter-Process Communication) technology, create a protocol on top of it for inter-application communication, produce a code library for the clients and, lastly, help them integrate it.

The Real Problem

The Creative Suite was only one version into being a suite, and the teams comprising it hewed closer to their roots as wholly independent entities than collaborators on Suite integration technology. I not only had to create a compelling, workable solution, but also persuade the lead architects of major applications that it was technically sound, *and* convince product management—both mine and that of Acrobat, Photoshop, InDesign and others—that it was worth the effort.

Getting Started

Step one, select which IPC to use. I researched all the IPC technologies available, wrote a white paper comparing and contrasting them, made a confident recommendation (named pipes) and then solicited feedback from the stakeholders. My proposal competed in part against an existing library used by Photoshop, and this caused some friction. So I chose to co-opt that code and build from it. It was not intended for the general purpose reuse I desired, so I eventually rewrote almost all of it, but I still feel that was the correct political and technical decision. It got me up and running quickly and earned the trust and faith of a potent client.

Another Protocol?

Did we really need another protocol? I had deep experience with several. I designed one of my own before (AcroTalk, used between Acrobat and CreatePDF.com). I worked with an outside professor to bring his SOAP package into Adobe. And I also designed

and co-wrote WebAccess, a C++ library for HTTP and SSL based on top of Tim Berners-Lee's original libwww. But none of the extant protocols was simple enough. Other architects would rightly be scared off by something as complex as HTTP or SOAP. So I patterned my new protocol, BridgeTalk, in the form of HTTP, but having only a few fundamental headers and no multiple verbs like GET/PUT/POST. That made the parsing very easy and flexible, since there was no requirement to respond to any particular action; all messages were simply a set of header values and the message body.

Control Issues

But what would the clients do with the messages? We could create a small set of calls and have all the clients integrate that support. But I was creating the scripting API for Bridge in ExtendScript, and so chose to build upon the existing scripting APIs of the Suite apps. The payload of a message would execute as a script in the target application's scripting environment and so could take full advantage of its entire API. I also defined a Cross-DOM API which was the minimum set of functions to be useful, like bringing the app into focus and opening a file.

Integration

I occasionally worked on ExtendScript with its creator, Michael Daumling, either adding or requesting features. So I chose to deliver BridgeTalk as part of the ExtendScript library. This made it easier to integrate because it wasn't "yet another library" for the clients to deal with. I also created an integration wiki and walked several engineers through the process, refining my guidance docs each time until eventually I was no longer needed and teams simply took care of it themselves.

Success!

It worked. Its lightweight, file-based app registration system became the canonical repository of "which Adobe apps were installed where." I added a notion of startup scripts which allowed launch-time configuration for BridgeTalk-based services. I also provided a way to remotely access scripting APIs so you could write scripts like this:

```
illustrator.open("louvre_*.ai");
illustrator.saveAll( "/tmp", "png");
photoshop.photomerge( "/tmp/louvre_*.png" );"
```

Eventually, almost every application at Adobe integrated it, although with significant security modifications required for consumer-use applications like Acrobat Reader and Flash.

Moving On

I did some knowledge transfer to Bernd Paradies, who took over from there. He designed and implemented BridgeTalk version 2.0, which I believe is the current one in use by the Suite.

Why This Story?

I believe this story shows that I can take an idea from cradle-to-grave (or—better than the grave—a new life in the next version). And this product started from a simple germ of a problem: How de we find out what apps are installed and switch focus between them? I feel I took that small charter and successfully solved the larger problem of how Adobe apps should communicate and control each other.

I also had to solve issues beyond the technical. Creating consensus amongst a wide range of technical experts doesn't always revolve around "correctness" of a technology proposal. They need to believe in the people and processes involved and be persuaded that their precious resources will be well spent.