WHAT ABOUT OSGI?

When we started this project in late 2004, the OSGi framework was just beginning to gain broader visibility—thanks largely to Eclipse's adoption of it. We looked at it briefly, but were put off by the lack of widely available knowledge, expertise, and guidance.

OSGi's purpose, though, is a perfect fit for the problems we faced. Supporting multiple deployment configurations with a common codebase, managing the dependencies among modules, activating them in the correct sequence...clearly solving the same problem.

I suppose the fact that we didn't use OSGi was partly a quirk of timing and partly our own reluctance to take on what we perceived as more technical risk. I usually come down on the side of "acquire and integrate" rather than "roll your own," but there seems to be a tipping point: lightly supported open source projects with weak communities are more of a risk than well-understood, widely adopted ones. Likewise, I tend to avoid quasi-open frameworks that are actually vendor consortia. The community they serve is usually the community of vendors, not the community of users.

It wasn't clear to us which camp OSGi would fall into. If we were doing the project today, I think we probably would use OSGi instead of rolling our own.

Kiosk-Style GUI

Studio associates are hired for their ability to work well with the camera and the families, especially children, not for their computer skills. At home, they might be Photoshop gurus, but in the studio, nobody expects them to become power users. In fact, during the busy season, a studio might bring on a number of seasonal associates. Consequently, fast ramp-up is critical.

One of the architects also served as our UI designer. He always had a clear vision of the interface, even if we didn't always agree on how much was feasible to implement. He wanted the user interface to be friendly and visible. There would be no menus. Users would interact with images through direct manipulation. Large, candy-coated buttons made all options visible. In short, the workstation should look like a kiosk.

That left the decision about what technology to use for the display itself.

One of our team made a survey of the Java rich UI technologies available, mainstream and fringe. We hoped to find a good declarative UI framework, something to help us avoid an endless slog through Swing tweaks. The results shocked us all.

In 2005, even after a decade of Java, two basic choices dominated the mainstream: XML hell or GUI builder spaghetti. The XML variants map more or less directly from Swing components to XML entities and attributes. This made no sense to us. GUI changes require a code release, whether the changes are implemented in straight Java code or in XML files. Why keep two