

Foreword by Jeff Norris

On June 10, 2003, I counted down the last seconds to the launch of the Spirit Mars Rover with the rest of the Mars Exploration Rover Mission team. For my team, it was a break from the final months of development and testing of Maestro, the software application that we would use to analyze the data received from the rover and plan its science activities once it arrived on Mars seven months later. One remarkable fact about launching something like Spirit is that the actual rover, or *payload*, is crammed into a tiny chamber at the very top of a huge rocket, called the *launch vehicle*. The launch vehicle is a fiendishly complicated system all on its own and its criticality can't be overestimated. Still, it's not really the point of the mission.

I believe that Maestro and every other rich client application can also be divided into a *payload* and a *launch vehicle*. An application's payload is the reason you wrote it in the first place. For Maestro, it's the part that deals with spacecraft telemetry, immersive visualization, and mission planning. For your program, it might be the part that manages a database of thousands of products at a dozen warehouses or the part that seamlessly deploys intricate Web sites to high-performance servers. In any case, the payload is the part of our programs that we daydream about in boring meetings and in the car on the way to work. It's the part that we love to work on.

Then there's the program's launch vehicle, which is everything else. Every application should be easy to use, which means it needs to provide things like a help system, progress updates for lengthy operations, and a way for the user to access and arrange views of data without getting overwhelmed. It needs dozens of little interface details like drag/drop, copy/paste, and undo/redo. Finally, the development team needs an architecture underneath it all that lets them add new features easily, deploy them quickly to their customers, and then update them later. The launch vehicle is critical to the success of your program, but its only purpose is to support your payload.

In June 2004, one year after launch and five months after Spirit and Opportunity arrived on Mars, my team turned our operations duties over to replacements and eagerly started working on the future of our software. When we looked back at the last version of Maestro, we realized that we had gotten our payload right, but our launch vehicle had some problems. A tight budget had forced us to leave out a lot of key things like copy/paste and a help system in order to finish payload features like image processing. We didn't have sufficient time to test things like software update, forcing us to eventually abandon it in favor of more arduous methods. We also reached the somewhat painful realization that we were simply not experts on making a highly usable application, or designing a flexible and sustainable development architecture. The launch vehicle is something that is extremely difficult to get right, and what we wanted most was just to focus on Maestro's payload.

The Eclipse RCP was the launch vehicle we were looking for.

Since adopting the RCP, our team has been able to retire thousands of lines of code from our old program in favor of features provided by the Eclipse RCP. Maestro greets its users with a welcome page and provides cheat sheets to walk them through the basics of the application. We use the Jobs application programming interface (API) to keep users informed when the application is performing lengthy operations and preference stores to allow them to adjust settings in the program. Perspectives let our users organize our many views and switch quickly between different mission operations tasks. We use the update manager to make sure our users are using the latest versions of our plug-ins. Our visualization and planning components are built on top of the Graphical Editing Framework (GEF), and we're planning to make heavy use of the Business Intelligence and Reporting Tools (BIRT) system to display reports on spacecraft resource usage. It's hard to find a part of our program that doesn't rely on some Eclipse capability and we're constantly discovering new technologies being developed in the Eclipse community that are enabling us to reach our goals more quickly.

In addition, we've used the modularity offered by the feature and plug-in architecture in the RCP to deliver highly specialized applications to our missions. In the past, our customers would receive a monolithic "core" application with a layer of adaptation that was designed to address their specific needs. This enabled us to reuse software between projects, but this strategy caused the core of the tool to grow larger and more unwieldy as each project added functions that the users needed. Dependencies between plug-ins in the Eclipse RCP are explicitly defined and enforced, so now we can safely pick and choose only those plug-ins that a particular customer needs rather than forcing everyone to use a monolithic "one size fits all" program. Maestro has become something of a

chameleon—different customers get exactly the functionality they want without the distraction of features they don't need.

The RCP has also become the centerpiece for a new consortium of operations software development teams within NASA called Ensemble. Multiple development teams at the Jet Propulsion Laboratory and Ames Research Center are using the RCP to develop and integrate their tools in a way that will enable more efficient mission operations. I think we will see more consortiums built around the RCP in the future as other kinds of organizations decide to pool their resources and share the responsibility for things that their programs have in common.

Choosing a framework for an application is a decision that must be handled with as much care as the selection of a rocket that will carry a spacecraft into orbit. Fortunately, you don't have to base your decision on testimonials or hype—this book offers you a “test flight” on the RCP. Perhaps you'll find that this launch vehicle will get your payload where it needs to go too.

—Jeff Norris
Supervisor, Planning Software Systems Group
Jet Propulsion Laboratory
California Institute of Technology
July 2005