Realtime publishers

The Essentials Series: Mainframe Application Modernization

Options for Modernizing Mainframe Applications

sponsored by

by Don Jones

Copyright Statement

Options for Modernizing Mainframe Applications

So how can we begin exposing our mainframe application's data and services outside the boundaries of the mainframe computer? I want to try to address the problem from three different angles: dealing with the hardware, dealing with the applications themselves, and dealing with the data directly.

Solution: Rehost

One option is to *rehost* your applications, something that's commonly done with applications written in a language such as COBOL. Rehosting lets you get rid of your mainframe hardware by moving the COBOL application to a non-mainframe host, which might be a UNIX or Windows computer, and might even be virtualized. Doing so gets the application running elsewhere, but it ultimately doesn't do much to change the monolithic nature of the application. Sure, you can cut back on mainframe hardware support costs, but that's often the only savings. The data in that application, and the services it provides, are still pretty trapped inside the application—you haven't extended it to any other portions of your enterprise. This solution is a good option if the *only* thing you need to do is eliminate the mainframe hardware, but it's not really an option for truly modernizing the application itself.

Solution: Recode

Software is malleable. With the right skills, tools, and time, software can be changed to do almost anything. So if your current mainframe application doesn't do what you need—recode it!

This is the exact direction my company took with our AS/400 for several years. We had a team of about a half-dozen dedicated RPG programmers. We acquired the source code for most of our enterprise applications (at great expense, by the way), and we started customizing them mercilessly. Once we'd acquired that on-staff skill set, "reprogram the AS/400" became the answer to nearly every IT-related challenge that came our way. When our distribution center needed to integrate with new distribution hardware, we reprogrammed the AS/400 to talk to that hardware's UNIX-based controllers. When the distribution center wanted to use wireless terminals to facilitate stock movements, we reprogrammed the AS/400 to provide the needed functionality. Our AS/400 programmers had lengthy wish lists from nearly every department of the company, and it seemed like new and changed AS/400 entry screens were released every day.

What we gave up, however, was the ability to easily use more off-the-shelf applications. We could never upgrade our enterprise mainframe applications when the software vendors released new versions because we'd lose our massive customizations. We were unable—well, not unable but *unwilling*—to purchase other off-the-shelf applications because we had all these on-staff programmers; shouldn't we be writing this stuff instead of buying it? We were *very* unwilling to purchase applications that ran on anything but the AS/400, simply because we wanted to get the most from our heavy investment in the mainframe. So we mentally limited our options to what our programmers could do, and we physically limited ourselves to whatever they could actually accomplish in their 10 hours a day.

Solution: Repackage

These days, one of the most intelligent—I think—solutions to the whole problem is *repackaging.* Simply put, you write some kind of wrapper around your existing mainframe application to expose portions (or even all) of the application through more modern, standard interfaces. Maybe you want to expose the mainframe application as a set of Web services, a bunch of .NET Framework components, or a set of Java objects.

This approach works well because it directly addresses the fundamental problem of mainframe applications. As I said in the first article of this guide:

> Monolithic mainframe applications, however, tend to intermix the user interface (UI), the business logic, and the data management layers of an application. The only way to ensure that the right data gets into the application is to enter it into the user interface. That's great for manual data-entry, but not so great when you begin looking for ways to leverage your data in other ways and to connect your data with other systems.

Repackaging, done properly, can actually turn *the entire mainframe application into a middle-tier component.* It essentially automates the use of the existing mainframe application, exposing data and accepting input through a Web service, Java object, or whatever. External applications see a .NET Framework component; that component is actually a sophisticated engine that manipulates the native mainframe application, taking advantage of its inbuilt business logic and everything. This technique typically requires *no changes on the mainframe,* meaning you can use prepackaged, off-the-shelf applications *with no changes,* and finally get your data *off* of the mainframe—all while keeping your mainframe investment completely intact.

Better yet, in many cases, you can turn the entire application into not a single exposed service but a *set* of exposed services, essentially componentizing the application without touching the application itself. This means a large, monolithic inventory management application might be able to provide an inventory-query service, a reordering service, and other services—each of which is an element of the original mainframe application.

Consider Figure 1, which represents a typical mainframe application: input screens for data entry and retrieval, including embedded business logic, and an on-mainframe data store of some kind. This figure represents a standard, monolithic mainframe application like the ones you are probably already using.

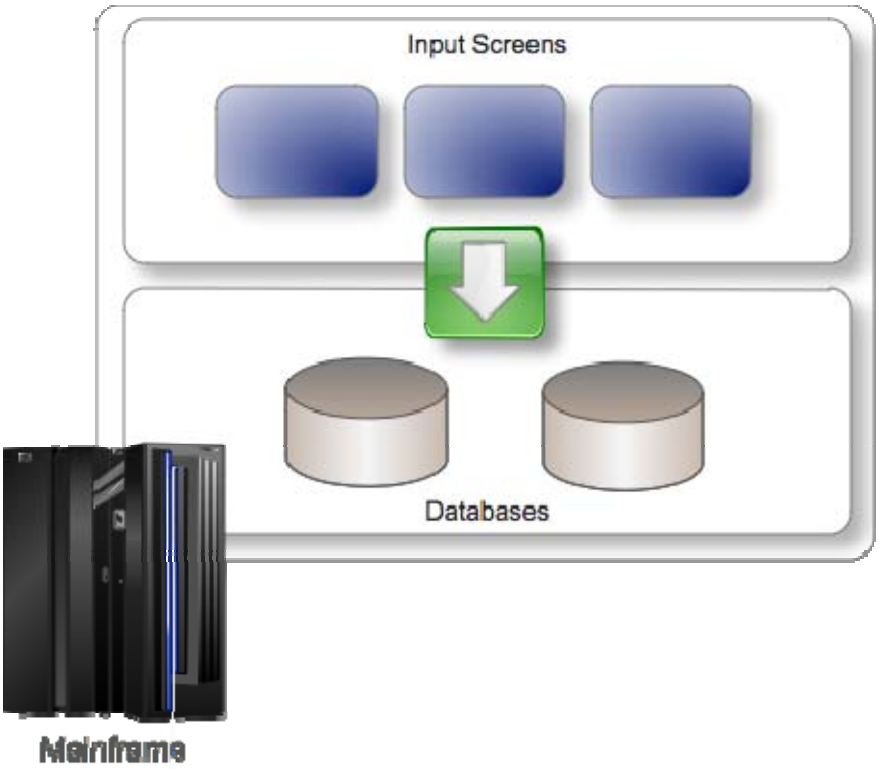

Figure 1: Monolithic mainframe application.

Because the application is monolithic, it can't be easily broken down into smaller components. That means, from the mainframe point of view, the entire application *is* the smallest component. However, you may well have external uses that just need the data from one or two screens, or just need to input data into a couple of screens. In other words, you've identified some element of the mainframe application that *could* be externally exposed as a distinct service. And you can do it: You simply have to repackage the application in such a way that the application can be used as-is—albeit in an automated, abstracted fashion—by other applications. Figure 2 illustrates this concept.

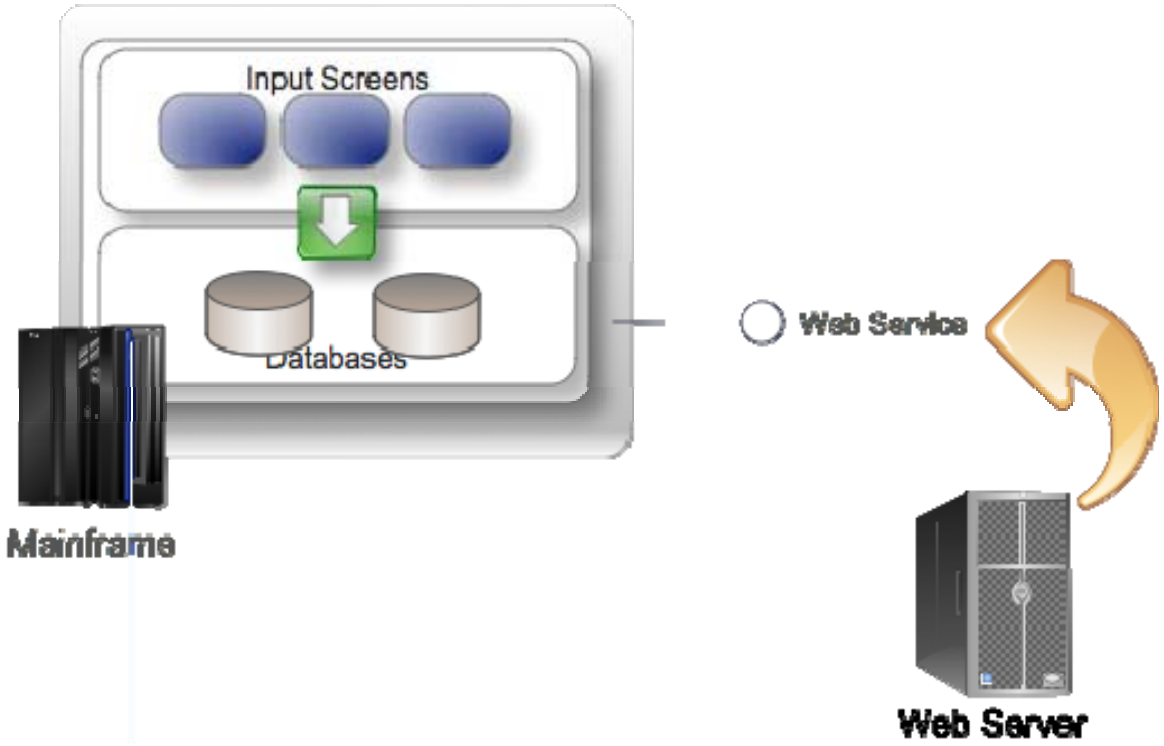

Figure 2: Repackaging a mainframe application.

This task requires some kind of repackaging engine. Essentially, that engine is taught how to automatically operate the mainframe application by reading its input screens and inserting data into the fields on those screens. The engine then exposes that data as a Web service (or Java object, or whatever) for non-mainframe applications to access—or as a *set* of Web services. You might very well *not* need to expose the entire mainframe application; you might only need to expose a portion of it, and treat that portion of the application as a component.

What's the benefit? The main benefit is that you still get to take advantage of the business logic that lives within the mainframe application—without modifying the application in any way.

Strategic or Tactical?

The general solution you choose is going to depend largely on your future plans. For example, choosing to rehost your application can help lower hardware costs, but it honestly won't do anything to change your ability to expose pieces of the application externally—it'll all still be locked up in a monolithic application. Rehosting is really a tactic designed to lower costs and driven by a larger strategic decision to *keep the application as-is.* You're not really modernizing anything; you're simply moving pieces around.

Recoding your mainframe applications is also a tactic driven by a larger strategic decision to—again—*keep the mainframe.* After all, why invest hours in recoding applications if you plan to move off the mainframe eventually? Recoding is also making a decision to continue operating in a mainframe-centric universe, where you want as much functionality as possible to live within the mainframe, and where you're willing to invest the necessary time and money to making that a possibility. Because recoding mainframe applications can be expensive, and because the skills needed to do so are frankly not in great supply, you may also be making a tacit decision to forgo business requirements that may *need* recoding, if the recoding skills, time, or money aren't available. In other words, you're happy letting the mainframe's technology drive, to a point, what your business can and cannot do.

Repackaging, however, doesn't lock you into anything. You can use it as a short-term tactical move: "We're going to repackage key portions of our mainframe application and use them elsewhere because we plan to migrate off the mainframe at some point." Repackaging also supports the longer-term strategic direction to stay *on* the mainframe: "We like our mainframe, we want to keep it, but we also want increased flexibility—which repackaging provides." Done properly (which I'll discuss in the next article), repackaging doesn't change anything *on* the mainframe, so it preserves your mainframe assets while minimizing or eliminating the need for costly custom programming on the mainframe. Repackaging, in other words, offers the most flexible kind of solution, and a solution that can fit many different strategic or tactical directions.

Repackaging Your Mainframe Application

In the final article of this guide, I'll look at the repackaging solution in more detail and discuss one approach that can either offer a long-term strategic direction or be utilized as more of a short-term, immediate tactical solution that helps support a particular long-term strategy.