Realtime publishers

The Essentials Series: Mainframe Application Modernization

Considerations for Injecting Life into Mainframe Applications

sponsored by

by Don Jones

Copyright Statement

Considerations for Injecting Life into Mainframe Applications

To quickly review the key points from the prior two articles in this guide:

- We need to be able to safely interact with portions of the mainframe application in non-mainframe applications
- We don't want to access the data directly, because then we're losing the application's business logic
- Repackaging allows us to build a wrapper around the entire mainframe application, or portions of it, making it accessible via Web services, .NET Framework, or Java

There are different techniques for accomplishing this repackaging, depending on exactly what kind of mainframe application you're dealing with.

Repackaging Techniques

These repackaging techniques all rely on the presence of some kind of purpose-built mainframe integration engine. The engine is what does the work of talking to the mainframe in the proper fashion, and exposing the resulting data to the outside world through whatever means you choose. These engines commonly use a variety of techniques, and I'll discuss the three primary ones here.

Host Integration

The easiest to understand technique is probably *host integration.* Using this technique, the integration engine literally operates the mainframe application in exactly the same way a human being would. The basic technique is as old as mainframes themselves, and is commonly called *screen scraping.* Essentially, you explain to the hosting engine how the mainframe application operates: where data is located on each screen, what type of data is expected for each input field on each screen, what keypresses move between screens, and so forth. This information is called a *model.* You then indicate which pieces of information and which capabilities you plan to expose to the outside world. The engine exposes that information as properties and methods (of a Web service, Java object, or .NET Framework component), which can be used by any compatible external application. Figure 1 illustrates the concept.

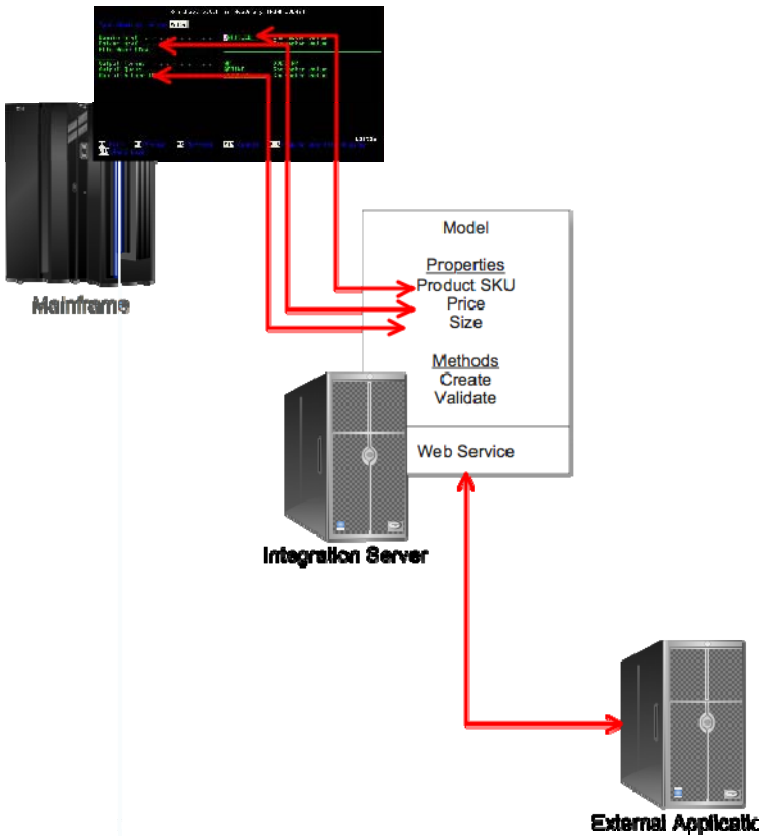

Figure 1: Modeling the mainframe application.

When an external application accesses data, the integration engine uses its model to determine where the mainframe application exposes that data. The engine manipulates the mainframe application to get to that point, reads the needed data from the screen, and then passes it to the external application. Integration engines commonly support a variety of terminal emulation protocols that enable them to talk to almost any mainframe, including IBM 3270, IBM 5250, VT/UNIX, HP700/92, and so on.

A great side benefit of this technique is that it's easy to manage change. If the underlying mainframe application changes—say, you upgrade a version—you don't need to modify your external applications. You simply update the integration engine's model so that it knows where to find all the data again. The engine thus serves as a kind of abstraction layer, helping to hide any complexities of the mainframe application and exposing the data in a straightforward, standardized, *modern* fashion.

The main benefit, though, is that you're getting the full capability of the entire application: its data, it business logic, and so forth. You're simply enabling things other than human beings to interact with selected portions of the application. In essence, you've taken a user interface (UI)-centric application and turned it into a sort of middle-tier component, exposed through modern architectures such as Java, COM, or .NET.

Bridge Integration

Some highly-standardized mainframe applications may be accessible through somewhat more sophisticated means. IBM's CICS software, for example, exposes a well-documented set of application maps called BMS maps. Rather than manipulating CICS' UI screens directly, an integration engine can tap into the underlying application through these maps via an IBM-provided bridge, as illustrated in Figure 2.

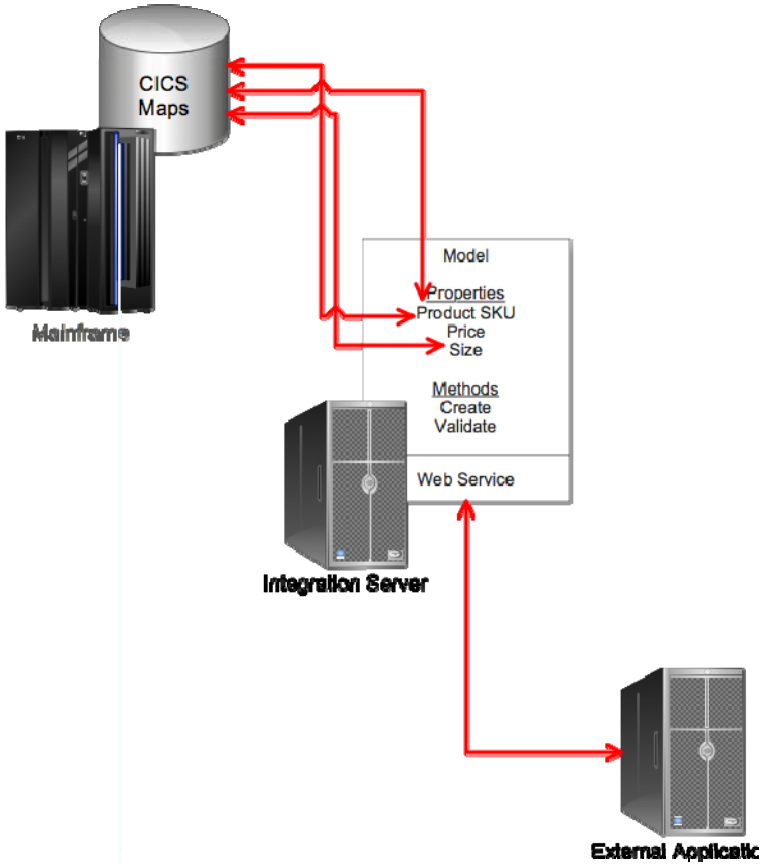

Figure 2: Tapping into CICS by using bridge integration.

> **Note**
>
> I've illustrated a "model" here, although it's important to note this isn't something you have to build directly. It's something the integration engine maintains on its own.

This technique is somewhat faster because the integration engine can talk directly to CICS application logic before the application renders a screen. The bridge, in essence, has direct access to COBOL fields and variables *behind* the column-level data on the screen. There's no need to manually build a model, and there's no dependency on the mainframe UI, so if that changes, you don't have any extra work to do. Best yet, the bridge is actually manipulating the COBOL code of CICS directly, which means all the application's business logic remains intact and usable, even though you're not going through the UI.

The end result, however, is identical: Portions of the mainframe application (or all of it, if needed) are exposed to external applications *with no changes to the mainframe.* You don't have a single line of extra code running on the mainframe to make this happen, so it's a low-impact, *very* fast way of integrating mainframe data into many external applications. As with host integration, you're taking a UI-centric application and transforming it into a set of middle-tier components.

Transaction Integration

Finally, an even more sophisticated technique can leverage the COM area of CICS and IMS applications because these applications are actually designed to support a certain degree of integration by external applications. As Figure 3 shows, the integration engine is simply mapping the applications' native integration protocols with more modern, standardized interfaces such as a Web service. This technique interacts with the mainframe application *below* its workflow and application logic levels, meaning you're far further "under the hood" and able to do things that the application might not normally allow, which is both good and bad.

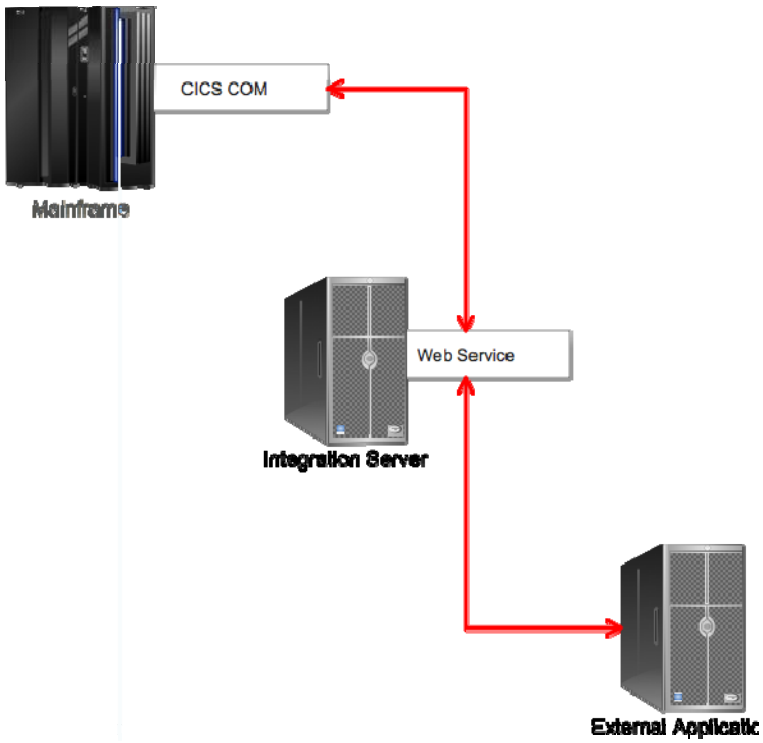

Figure 3: Transaction integration with a CICS or IMS application.

This technique offers a lot more flexibility than the others because you're working with the data at a low level. However, this isn't without risk: Because you're bypassing higher levels of code, you'll have to ensure that you don't break things like data dependencies. So the additional flexibility of this technique is offset by the greater responsibility you take on yourself.

Keep Your Mainframe—and Gain Agility

There are countless business reasons for getting your mainframe application services *out* of the mainframe. You can implement new business processes and new business capabilities and help keep your organization as agile as possible, ready to adapt to changes rapidly. Repackaging mainframe applications through the use of an integration engine that uses the techniques I've discussed in this guide, offers the fastest way to connect your mainframe applications to non-mainframe applications, using modern software interfaces such as Web services, .NET Framework, and Java. You essentially turn entire mainframe applications into sets of middle-tier components, which can be utilized by modern software built on standardized platforms, protocols, and techniques.

Best of all, these techniques require *no* changes to your mainframe hardware or its applications. This helps reduce the skills required to implement these techniques, helps reduce the risk to your critical data and applications, and allows you to leverage the existing business logic that's wrapped up in your mainframe applications.

There's no doubt that mainframe applications can play a major, positive role in the modern business world—with the right integration.