David Chappell
& Associates

INTEGRATION SOFTWARE: BUILD OR BUY?

DAVID CHAPPELL

MARCH 2010

SPONSORED BY MICROSOFT CORPORATION

No modern organization uses only one application. A single software solution just isn't enough to address the diverse needs of most firms, and it never will be. Also, because a firm's applications frequently need to talk to one another, they must be interconnected in various ways. There's no way around this—it's an unavoidable business reality.

Yet how should you connect applications? Is the best approach to buy specialized software to tie them together? Or does it make more sense to build what you need yourself?

The answer isn't obvious. In fact, the right choice depends on the problem to be solved. Understanding how to make this choice requires looking at the difference between simple connectivity and true integration. It also requires understanding exactly what functionality is required, then working out the pros and cons of creating it yourself. Once you've done this, making a good decision gets easier.

CONNECTIVITY VS. INTEGRATION

Connectivity is a straightforward idea. It just means connecting two applications, letting them communicate directly. For example, suppose a firm wishes to let its customer relationship management (CRM) application submit new orders directly to its manufacturing application. The goal might be to move orders more quickly from the salespeople to manufacturing or to reduce the number of errors in submission or something else. Whatever the reason, the solution requires nothing more than a direct connection between two applications. Figure 1 shows a simple picture of this scenario.

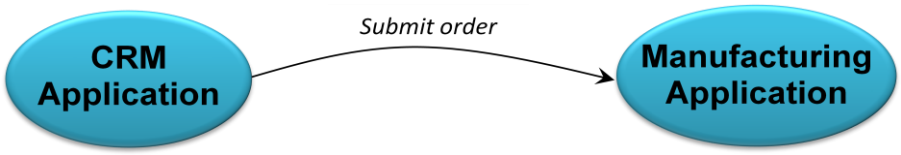

Figure 1: Connectivity means directly connecting two applications.

Solving a connectivity challenge like this typically requires addressing two problems:

- The applications must be able to interact in some way. They might communicate via SOAP, for example, or maybe they can be connected using a message queuing technology. The solution might even use a simple approach like writing to and reading from a shared file.
- The data exchanged must be translated between the different formats used by the two applications. In the example shown in Figure 1, for instance, the CRM application probably represents an order's information differently than the manufacturing application. This problem might be addressed by translating data into and out of a common XML representation or in some other way.

Solving these two problems usually isn't terribly difficult, so competent developers can create and maintain effective solutions for connectivity. In cases like these, specialized software probably isn't necessary.

Make the scenario a bit more complex, though, and the story changes. Suppose, for example, that the interaction shown in Figure 1 is part of a larger business process, and that the firm has decided to automate this entire process. The result might look something like Figure 2.

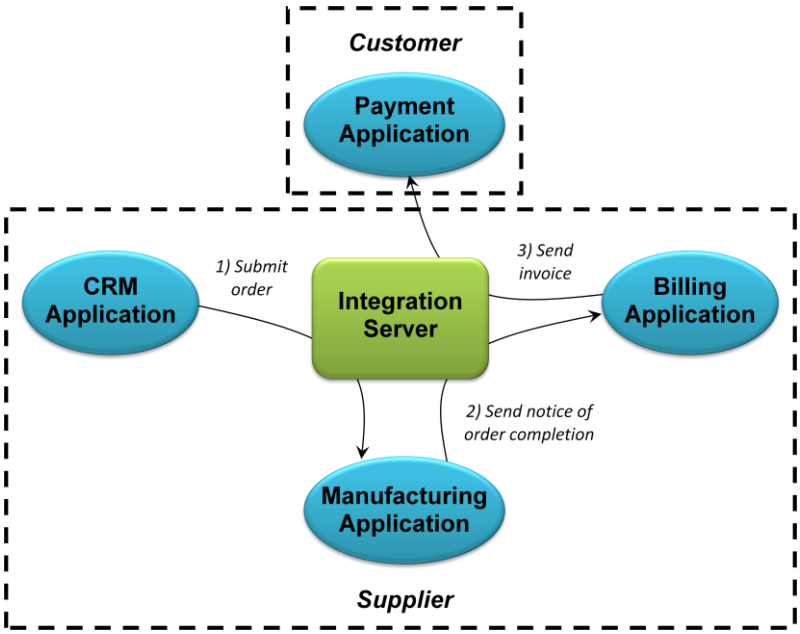

Figure 2: Integration relies on an intermediary to connect multiple applications, such as when a complete business process is automated.

In this example, the goal is to automate a multi-step business process spanning two organizations. To begin, the CRM application submits a new order to the manufacturing application (step 1), much like the previous example. Rather than being sent directly to the manufacturing application, however, the order first goes to an *integration server*. This specialized integration software then sends the order to the manufacturing application. Once the order is complete, the manufacturing application interacts with the integration server again, which notifies the billing application of the order's completion (step 2). The billing application can then send an invoice to a payment application in the customer's organization (step 3). Once again, this interaction relies on the integration server—the two applications don't communicate directly.

Automating this kind of multi-step business process, one that involves several applications across two organizations, is a common example of integration. Automated processes are typically faster, more accurate, and less expensive to operate, and so making this kind of investment can certainly be worthwhile. The technical challenge is more complex, however—this is integration, not just connectivity—and so are the problems to be solved.

Those problems include the following:

- Data must now be transferred between multiple applications, each of which might support a different communication mechanism.
- The format of that data must be translated across several applications, not just two. If automating the process requires communicating between different companies, as with step 3 in Figure 2, this business-to-business (B2B) communication likely requires conforming to a standard (and potentially complex) data format such as electronic data interchange (EDI).

- There must be a way to create and execute the logic that drives the entire business process. Because this logic isn't part of any of the applications, it needs a server to run in. And because automating all (or even a large share of) a business process is likely to make the integration software mission critical, this server must be reliable and scale well.

- Effective tools for creating process logic, data translations, and more must be provided. This is especially true when the process, the communication mechanisms, or the application data formats will change, as good tools can make these changes significantly easier to do. Management tools are also essential, since integration solutions require management like any other application.

Supporting integration is harder than providing basic connectivity. This reality is why integration servers exist.

EXAMINING INTEGRATION SERVERS

Several vendors today offer integration servers. Figure 3 shows some of the components of a typical product.

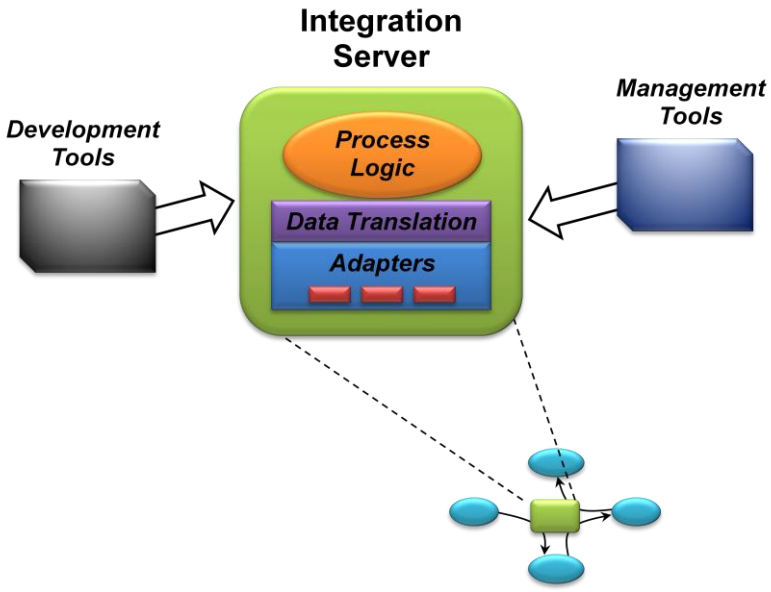

Figure 3: An integration server can provide adapters, support for data translation, a place to run process logic, tools for development and management, and more.

An integration server's components are designed to address the problems just described. As the figure shows, those components commonly include:

- *Adapters* that provide pre-defined software for connecting to various applications in diverse ways. One adapter might use SOAP-based communication, for example, while another allows interaction via message queues.

- *Data translation* services that provide a standard approach for mapping between the data representations used by different applications. Among other things, this can include support for EDI.

- *Process logic* support, offering a scalable, reliable execution environment for the logic that drives integration.

- *Development tools* for creating process logic and data translations along with *management tools* for managing an integration solution.

As shown earlier, an integration server acts as an intermediary between the applications being connected. Rather than communicating directly, all interactions now go through this server. This approach has several advantages:

- It lets each application use only one communication mechanism, then rely on the integration server to communicate with other applications. The alternative, connecting an application directly to all of the other software it interacts with, can require each application to support multiple communication mechanisms.

- It allows centralizing data translation in the integration server rather than requiring any of the applications to deal with this problem. This centralization also makes it possible for developers to use one set of tools to create and maintain the necessary mappings between different data formats.

- It provides an external home for the logic that runs the business process. Because the integration server is separate from any application, it can be made as scalable and reliable as necessary.

- It isolates each application from changes in the others. Without this, a change in the data format or communication mechanism of one application requires changes in all of the applications it talks to. With an integration server in between, a change to any application need affect only the integration server.

CONNECTIVITY OR INTEGRATION?

How can you tell if the problem you're facing is connectivity or integration? Answering this question can be hard. You might be faced with what looks like a connectivity problem, where all you're asked to do is connect two applications. Yet how can you be sure that this apparently simple problem won't eventually morph into a more complex integration scenario?

For example, the connectivity problem shown in Figure 1 was actually part of a larger business process, something that's true of many connectivity problems. If all you'll ever be asked to do is provide communication between these two applications, treating this purely as a request for connectivity makes sense: It's a cheaper and faster approach. But if a request to connect these two applications is in fact a precursor to a demand for broader process automation, you'll be better off taking an integration approach from the beginning.

Even if a single project doesn't justify the investment in an integration server, using one still might be the right choice from a longer-term perspective. A spaghetti of point-to-point connections, the bane of many organizations, doesn't happen overnight—it's the accretion of multiple connectivity projects. Once a chaotic set of independent connections is allowed to take root and grow, it can be hard to get rid of. And trying to create a unified integration from a set of existing point-to-point solutions is likely to be more painful than biting the integration bullet in the first place.

BUYING AN INTEGRATION SERVER VS. BUILDING YOUR OWN

Determining whether a problem needs connectivity or integration can be hard. But if you've decided that integration is required, prepare to spend some money. For true integration scenarios, making an investment in integration software is all but unavoidable. The question remains, however: Should you buy that software or build it yourself?

In general, answering this question is easy. Assuming the price is reasonable, you're usually better off buying an integration server rather than building one yourself. Here's why.

First, the problems to be solved aren't simple. As just described, integration requires addressing a list of challenging issues. Creating effective adapters for communicating with diverse applications is a significant project, as is building a range of data translations. For a user organization to write all of this itself is no small task. (This is especially true in B2B scenarios that use EDI.) Similarly, creating a scalable, reliable, and secure environment for running process logic is hard. And while graphical tools for defining processes can make creating and maintaining those processes faster and less expensive, building those tools also isn't easy. All of these areas require specialized skills that integration vendors have but most other firms don't.

Another reason for buying an integration server rather than building your own is that your internal people are unlikely to create a truly manageable solution. While the ability to manage an integration server might not matter much to its initial developers, it plays an important part in the total cost of ownership for this software. An effective integration server always includes good management tools, providing built-in support for this area.

Yet another issue is that the true investment in building your own integration software includes the long-term cost of maintaining it. This means not just the hours your staff spends keeping the code up to date, but also the resources you devote to keeping people with the right skills. Integration software is a specialized area, and hiring and retaining people with the knowledge to build and maintain this technology isn't cheap. While buying an integration server means you'll need to maintain people on staff (or as close consultants) who are expert in the integration product you choose, this is a less specialized skill set than what's required to create that integration server.

One more concern with writing your own integration software is the time it takes. Assuming your integration project has business value—and if it doesn't, why are you doing it?—getting the solution up and running as soon as possible makes sense. Buying an integration server, even if there's a learning curve for using it, will be faster than writing one yourself. Depending on the project's business value, it's even possible that the money you save on integration server license fees gets eaten up by the business cost of delaying the project.

A final issue is the opportunity cost. Why should a bank or retailer or manufacturing firm or government agency invest its resources in creating its own integration server? Organizations ought to develop custom software only when doing so provides a competitive advantage. Spending development resources creating infrastructure that's widely available from vendors is almost certainly less valuable than devoting those same resources to projects that provide business differentiation.

Just as most organizations don't build their own operating systems or database servers, a typical firm shouldn't build its own integration server. In the overwhelming majority of cases, the best solution to a true integration problem will be built on a commercial product.

CONCLUSION

If the problem you're facing is just to provide straightforward connectivity, you probably don't need an integration server. Think hard, though. Will what looks look a connectivity project today grow into an integration project tomorrow? Will there be multiple connectivity projects that could benefit from a more cohesive approach? And will the solution you create change over time? All of these are reasons to consider taking an integration approach.

When a solution does require integration, you'll generally be better off buying an integration server. While it's possible to build your own, the economics of purchasing a product are likely to be more attractive. If you need to solve a real integration problem, expect a commercial integration server to save you time, money, and headaches.

ABOUT THE AUTHOR

David Chappell is Principal of Chappell & Associates (www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technology.