Realtime publishers

Solutions for Automating IT Job Scheduling

Greg Shields

sponsored by

Advanced Systems Concepts, Inc.

Copyright Statement

Chapter 2: Seven Use Cases for Automating IT Job Scheduling

Find me the business that runs atop a single application—a single instance!—and I'll show you a business that doesn't need IT job scheduling. Everyone else probably does.

In fact, most data centers have far more. Your average midsize data center runs applications for handling its databases, along with middleware systems for processing the data. That data center requires servers and protocols for staging of data in and out of the organization. Applications run atop client/server operating systems (OSs) and mainframes, servers, and perhaps even a few desktops. All of these elements need to communicate with each other, many don't share the same OS, and all suffer under the management complexities brought about by product-specific toolsets.

Today's IT technologies are fantastic in the business processes they automate, but rare are two that seamlessly talk with each other. Rarer still is the IT product that is superior all by itself in creating and scheduling workflows that meet business requirements. Needed to integrate activities among disparate technologies is a central solution that can interact with each at once.

An IT job scheduling solution is that *Rosetta Stone* between different platforms, OSs, and applications. It is intended to be the data center's solution for converting raw technology into business processes. In this book, I hope to show you how to incorporate such a solution into your own business.

You've already experienced a taste of how an IT job scheduling solution might work. Chapter 1 was constructed to help you recognize that job scheduling is a service your IT organization probably needs. That said, Chapter 1's discussion intentionally stayed at a high level. You haven't yet explored deeply the features and capabilities such a solution might bring.

You won't get that deep dive in this chapter either. That's because I've found that the best explanation of IT job scheduling requires first a look at the problems it intends to solve. Once you understand where it fits, you'll then appreciate the logic behind its behaviors. It is my hope that by the conclusion of Chapter 1 you began nodding your head, affirming that this purported solution is something your data center desperately needs.

Seven Stories in IT Job Scheduling

My task now is to further enlighten you with a series of ideas to help you find that best fit. These ideas will take place in the form of a seven use cases; essentially, seven little "stories" about issues that have been resolved—or made easier—through the incorporation of a job scheduling solution. These stories themselves will be mostly fictitious but are based on real events and real problems. I'll use faux names to keep the narrative interesting.

There's an important point here. Even if some portion of these stories is made up, you should find that the problems and solutions in each aren't far from those you're experiencing.

Story #1: Five Administrators—Five Scheduled Tasks Libraries

The first of these stories has nothing to do with a customer-facing solution. Neither is it directly related to a line-of-business application. Rather, the first of these stories starts simple. It explains the administrative situation at Company A, a mature company with a procedurally-immature IT organization. Lacking many centralized processes, operating with marginally-effective change and configuration control, and managed by five different administrators, Company A's data center is a mish-mash of fiefdoms and technology silos. Problem is, these fiefdoms need to communicate with each other, even if their managing IT administrators won't.

John, Bob, Jane, Sara, and Jim are those five IT administrators (see Figure 2.1). Each is responsible for some portion of the data center infrastructure, with each having some overlap of responsibilities. To accomplish administration, they've created scripts, tasks, and packages that keep the individual business workflows running. Those automations indeed enact change on servers and get data moved from system to system but with no interconnection of intelligence.

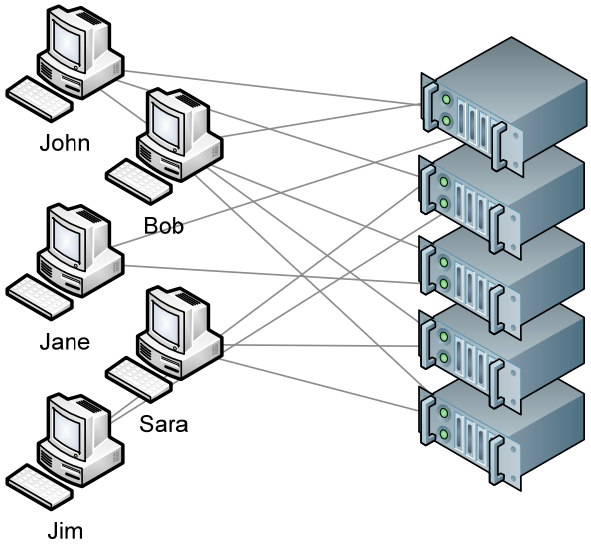

Figure 2.1: An interconnection of automations.

Figure 2.1 explains this problem in graphical form. In it, you can see that individual automations are sourced without considering their context. If John creates a job, his information cannot be based on instrumentation that is gained through another created by Sara. As a result, there is no way to orchestrate the activities between each individual, no way to schedule activities so that they do not conflict, and no way to base information or scheduling from one off of the results of another.

A much better solution is in aggregating these five people's automations into a single and centralized solution. Through that single solution, each administrator's jobs can be seen by the others. The jobs of each person can also be aligned with the needs of the others to ensure resources aren't oversubscribed. Additionally, because jobs are collocated in a single location, information and instrumentation from any automation can be used to drive other automations—or feed into their future scheduling.

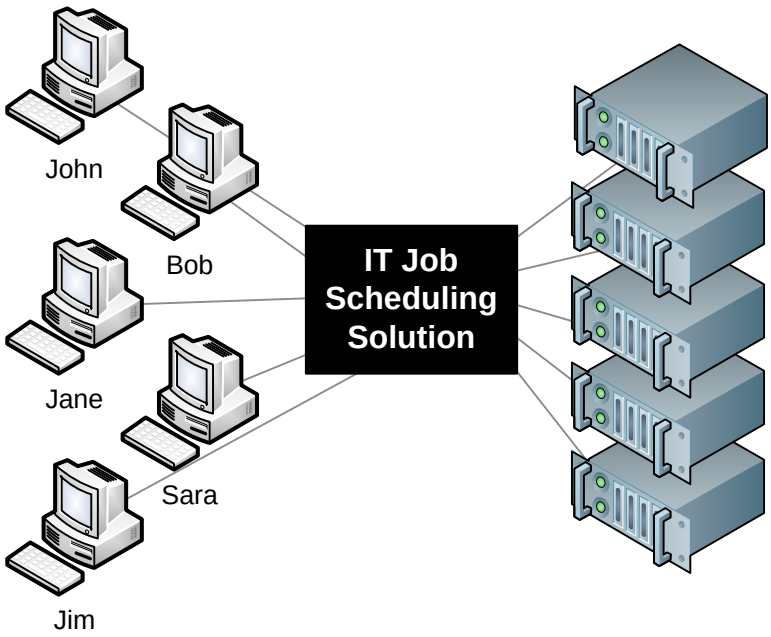

Figure 2.2: Sourcing automations through an IT job scheduling solution.

In short, even if your automations are administrative in nature, an IT job scheduling solution can bring substantial benefit.

Story #2: Consolidating Tasks from Every Application

Yet an IT job scheduling solution isn't solely about its actors. In fact, in many ways, the actors can be one of that solution's least-important impacts. An IT job scheduling solution really *has more do with the data in a data center*. That's why the second story in this chapter deals with the different applications that are used by Company B.

Different from the IT administration example told in the previous story, Company B's story centers around their line-of-business (LOB) application. That LOB application is comprised of several components, each of which is represented in Figure 2.3. Transactions among these systems occur through a carefully choreographed set of tasks, jobs, packages, and workflows. As you can probably imagine, the system in aggregate crosses Windows and UNIX boundaries, and includes multiple database management systems and even a bit of middleware. It is the classic business service.

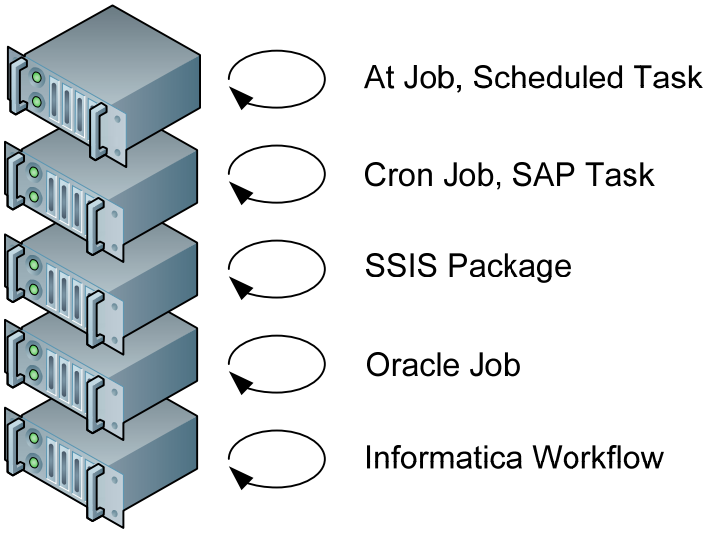

Figure 2.3: Individual schedulers for each individual application.

All of these individual components enable the functionality of the LOB application. But all also leverage their own built-in toolsets for scheduling activities: The SQL server runs its SSIS packages, the Linux SAP server runs its own tasks and cron jobs, and even the Informatica server enacts change through its workflows.

You are correct in assuming that this one-scheduler-per-component configuration can indeed work for many systems. Data and actions that occur inside Informatica can be based on wall-clock time or other schedule characteristics. The SQL database can run its SSIS packages based on its own settings, and so on. However, like the actors in the first story, this environment is likely to experience problems as individual system activities conflict with those on other systems.

Contrast this situation to the superiority in design one gets through job consolidation. In Figure 2.4, the individual task schedulers in that same LOB application have been replaced by a single and centralized IT job scheduling solution. This is possible because, as I mentioned in Chapter 1, a primary benefit of such a solution is its ability to speak the language of every application in the business service.

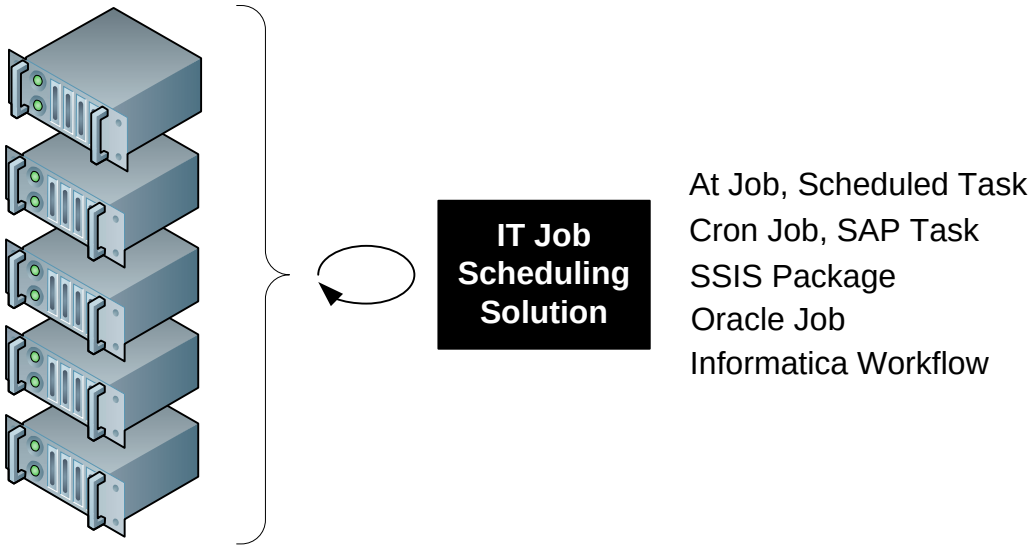

Figure 2.4: Consolidating tasks across applications.

With that centralization of data and actions comes an enhancement to job scheduling, based on results or data in other jobs. This chapter's sixth story will explain in greater detail how triggering capability dramatically improves service performance; but know here that centralization of scheduling brings to bear greater instrumentation about the health of jobs across the entire application infrastructure.

Story #3: The Script that Wasn't a Job

John is an Oracle DBA who has been with Company C since the very beginning. In his role as database administrator, he built the company's business system infrastructure nearly from the ground up. As a result, he understands those systems inside and out: He has tuned the system over time to improve its performance and weed out non-optimizations. He's built numerous scripts and other automations that gather data, translate it, process it between application components, and present reports for review by stakeholders. That system is critical to the company. It provides important revenue data for its sales teams and executives. It also means a lot to John.

Then one day the company grew. Substantially. Overnight. Acquiring a completely new line of business, Company C suddenly found its internal IT systems insufficient to handle the new reality of work and its associated data needs.

John was approached shortly after the merger by some very important people in the now larger company. Those people recognized his strengths in creating and managing the original revenue system that brought much value to the smaller company. They wanted another system, "…just like the first one, but this time for selling sprockets instead of cogs."

Graciously accepting the offer to improve the company and fortify his resume, John immediately realized that simply replicating the original system would not be a trivial task. Although his scripts absolutely did everything requested of them in the original system, they were also hard-coded into that original system. Its database architecture was designed to deal with cogs. His transformations were cog-based in nature. Even the server names and script names were hard-coded into each individual script, task, and package. Worse yet, there were *hundreds* of tiny automations spread everywhere.

Translating even a simple database job from cogs to sprockets, like what you see graphically in Figure 2.5, would take months of detective work, recoding, and regression testing. John was in for a great deal of work, and the result might not be as seamlessly valuable as his original system.

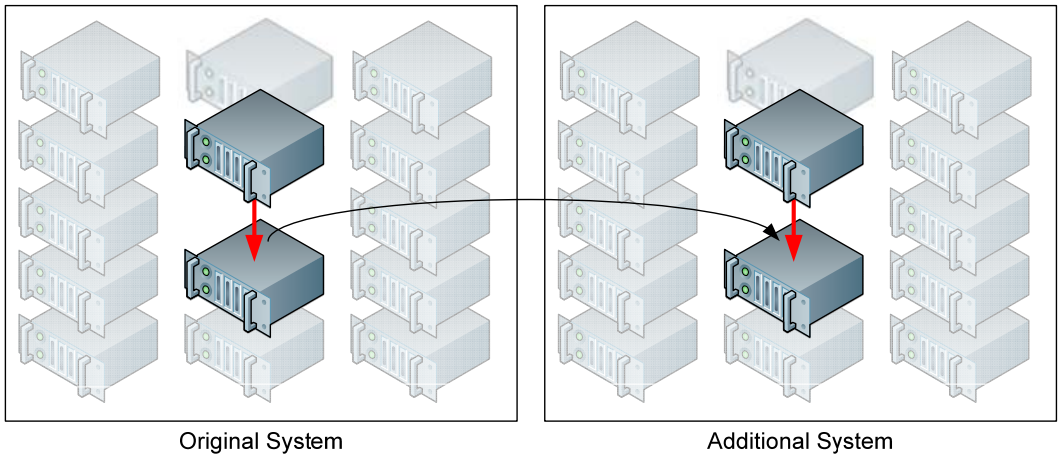

Figure 2.5: Replicating an automation to a completely new system.

Had John's scripts, tasks, and packages instead been objects within a central job management system (see Figure 2.6), this new company need might not be fraught with so much risk.

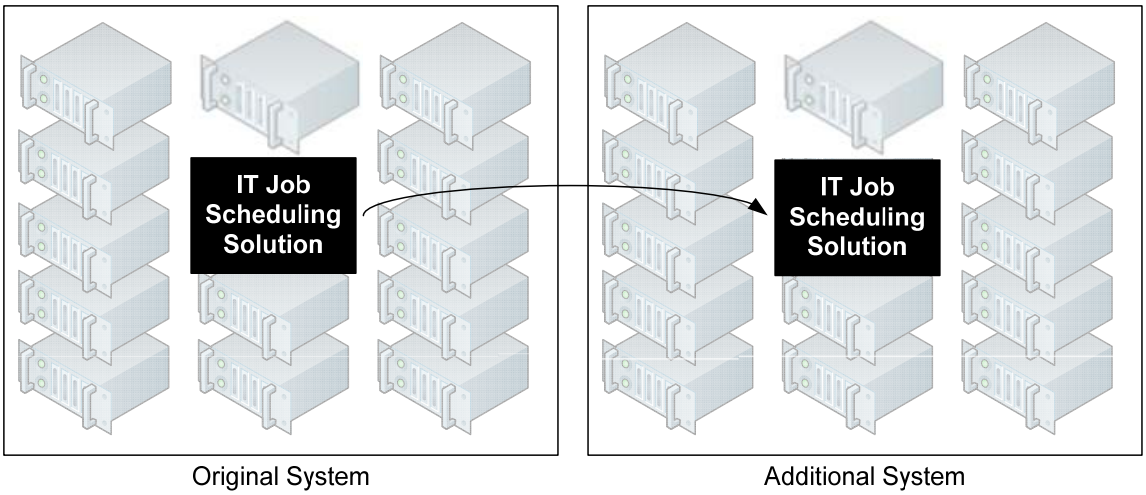

Figure 2.6: Objects remain objects as they're translated into a new system.

Recall Chapter 1's conversation about how good IT jobs are those that are coded for reusability. IT plans are then constructed out of individual job objects to enact change. In good jobs, variables are used to abstract things like server names and script names—sprockets and cogs, if you will—so that entire plans can be re-baselined to new systems with a minimum of detective work, recoding, and regression testing. An IT job scheduling solution takes the risk out of business expansion, giving IT the flexibility to augment services as the business needs.

Story #4: When Gathering Data Is More than Gathering Data

Company D's current situation is a product of its own success. Starting as a small organization with a single mission, their focus has changed and evolved over the years as lines of business come and go. Indeed, even whole businesses have been grafted on and later spun off as the winds shift in Company D's industry.

As a result, Company D suffers under many of the problems you would associate with any classic enterprise company. It has thousands of applications under management, some of which are used by only a very few people. Some homegrown solutions only remain because they were coded years ago to solve a specific problem for a specific need that has not changed.

Being a company that is more like the summation of lots of little companies, these business applications are nowhere near homogeneous. One budgeting application might store its data in SQL, another in Oracle, a third in some obscure database language spoken only by IT professionals long past retirement. Tying these applications and their data together is a big job with big consequences to the organization.

Many enterprise organizations that rely on disparate databases leverage Business Intelligence (BI) solutions like Crystal Reports. These BI solutions aggregate information across the different architectures. Using BI tools like Crystal Reports, data in an Oracle format can be compared and calculated against data in a SQL format, and so on. These tools come equipped with rich integrations, enabling them to interconnect nearly all database formats all at once (see Figure 2.7). Company D uses Crystal Reports to gather budgetary data across business units and individual project teams.

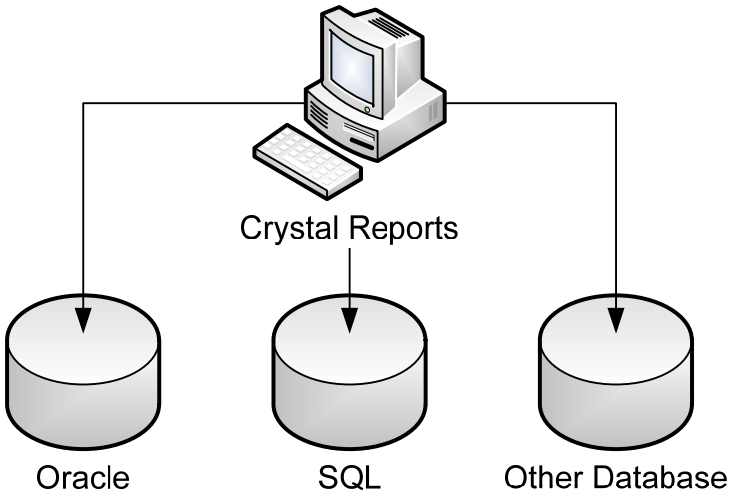

Figure 2.7: Connecting solutions like Crystal Reports to multiple databases.

Yet Figure 2.7 doesn't fully show the reality of how Company D's data is generated. Before that data ever becomes something tangible that can be ingested into a BI solution, it starts its life inside any number of down-level systems. Figure 2.8 shows just a few of those underlying systems, all of which integrate to create the kind of data a BI solution desires to manipulate.

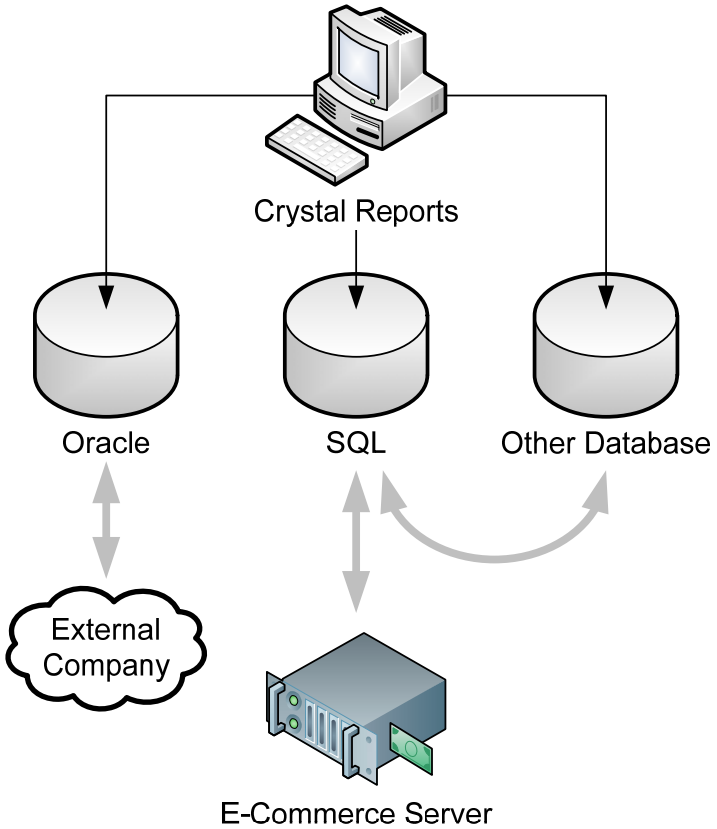

Figure 2.8: Underlying jobs make Business Intelligence data usable.

Notice in Figure 2.8 how a portion of the first business unit's information comes from a partner company external to the data center. The second and third business units have projects in combination that require orchestration and synchronization between databases. The second business unit has further integrations into an e-commerce server in order to gather a full picture of budget levels.

BI solutions can indeed present a more-unified view of data across different platforms, but they do not provide a mechanism to unify transactions between down-level systems. For Company D, creating that unified workflow lies within the realm of an enterprise IT job scheduling system. If gathering data is more complicated than simply gathering data, a job scheduling system ties together the entire system to accomplish what you really need.

Story #5: Controlling Data Transfer as a Workflow

Company E had a big problem not long ago when they began extending customer services onto the Internet. They quickly found out that you can indeed interact with customers there, but creating a holistic system that gets customer data in all its various types into the hands of the right person isn't as easy as it looks.

What Company E found out is that dealing with customers over the Internet automatically creates a lot of data. That data arrives in various formats, with each format requiring a different mechanism for handling.

You can see a graphical representation of Company E's story in Figure 2.9. Internet customers that desired services interacted primarily through a Web server. Inside that Web server was contained the requisite logic to inform customers about products, and interact with them as purchases are made. Interesting about Company E is that their system involved two-way communication with customers not only via the Web site and email but also in the transfer of data files.

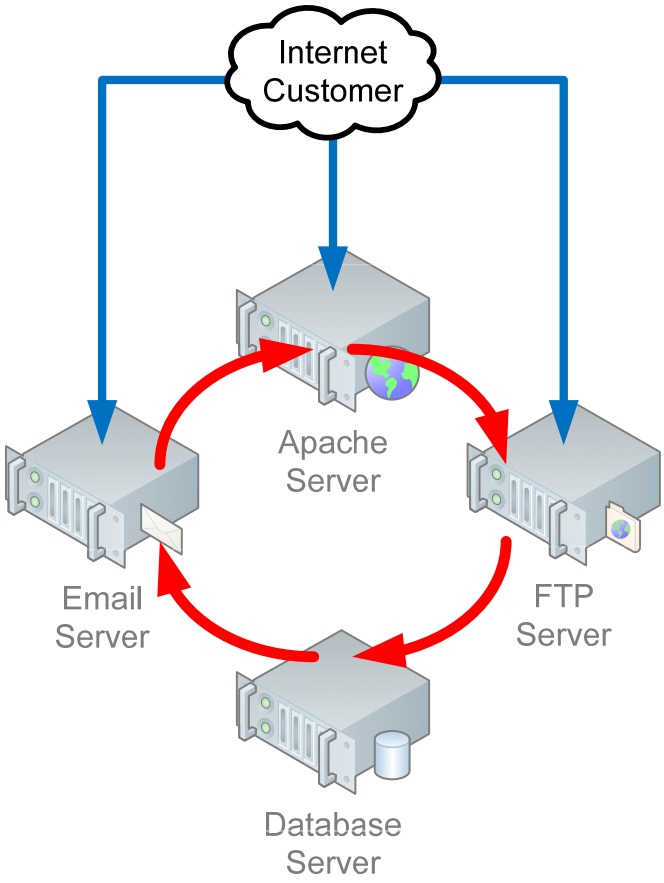

Figure 2.9: Different data formats require different data handling.

Data files, as you know, are much different than XML files in Web transactions or emails back and forth through an email server. They're larger, they can come in many different formats which create particular issues when you're working with unstructured customers, and the management tools to work with them don't necessarily integrate well into other formats and workflows.

Company E needed a multipart mechanism to solve their formatting workflow problem. They needed to recognize when an order was placed, generate an FTP URL for the user to upload data, move that uploaded data from a low-security FTP server to high-security database server, and finally notify the user when the transaction was complete. Adding to the complexity, those exact same steps were required in reverse at the time the order was fulfilled.

You can imagine the protocols and file formats at play here: XML, SMTP, FTP and SFTP, along with a little SSH and SOAP to tie the pieces together, just like you see in Figure 2.10. Complex needs like Company E's require data transfer handling that can support the recognition that files have been downloaded. Such handling can either monitor for a file's presence or use event- or message-based notification. An IT job scheduling solution wraps file transfer logic into the larger workflow, enabling XML to trigger SSH, to fire off SMTP, and finally to invoke SOAP at the point the application requires.

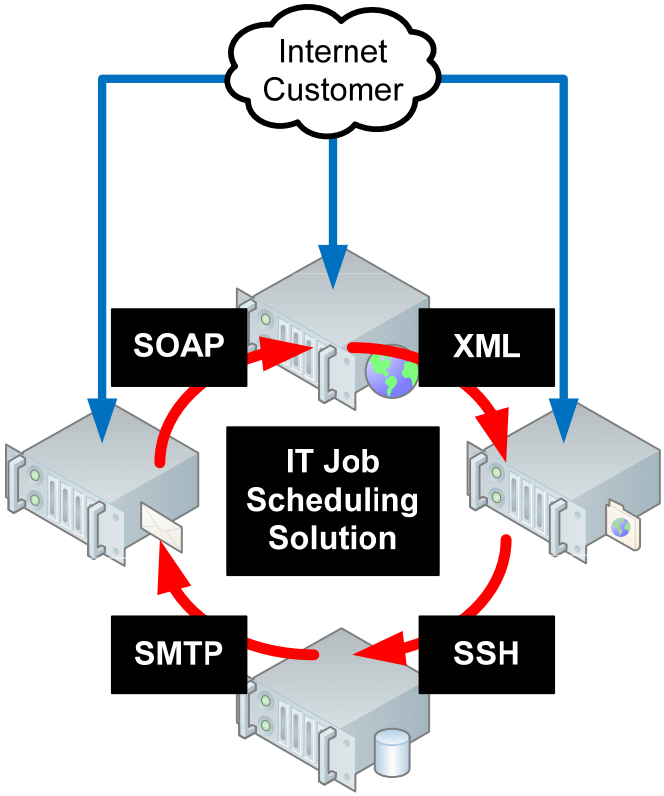

Figure 2.10: Multiple protocols at play, each with its own management.

For complex file transfer needs, those that must exist within a business workflow, IT job scheduling solutions solve the problem without resorting to low-level development.

Story #6: Complex Jobs and the Need for Triggers

This chapter's sixth story brings me back to the one I started in Chapter 1. There I explained some of the introductory pieces in *The Project That Would Change Everything*. I also explained how an IT job scheduling system was very quickly identified as the only class of solution that could enable the kind of functionality our complex project needed. Let me tell you a little more in that story.

Our determination happened shortly after whiteboarding the various components we knew the project needed. Figure 2.11 shows a re-enactment of that whiteboarding activity, with only a few of its myriad interconnecting magic marker lines in place. My team knew that when external company data arrived on the FTP server, transferring that data to the SQL server must occur in as close to real-time as possible.

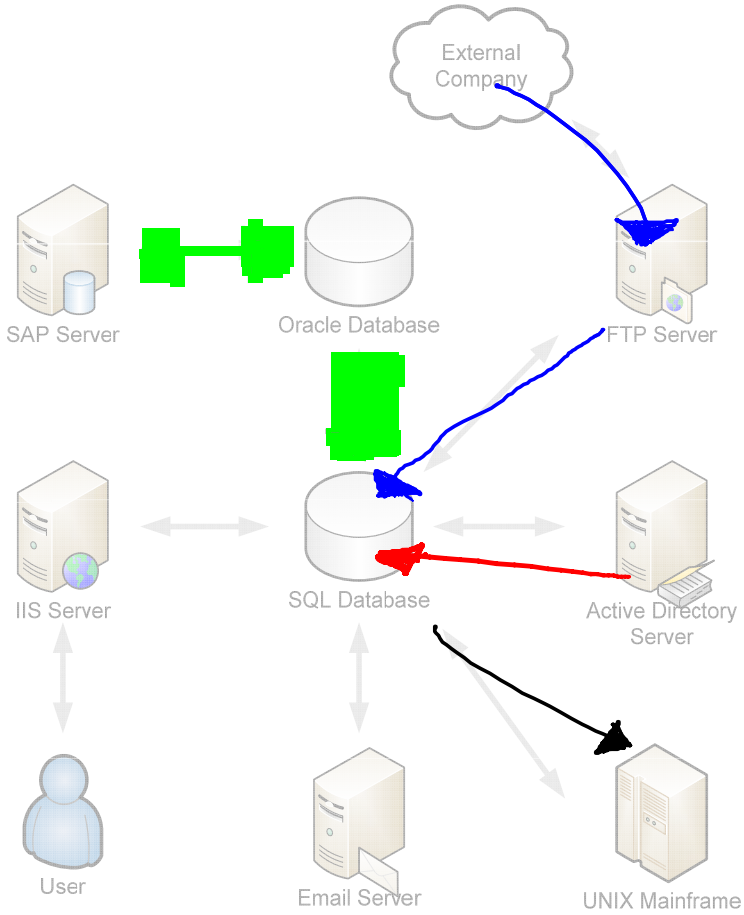

Figure 2.11: Whiteboarding the triggers for The Project.

Fulfilling that requirement with traditional FTP alone created a fantastically problematic design. The idea of constructing an always-on FTP session that constantly swept for new data was a ridiculous notion. And it wasn't a good idea for security. Needed was some kind of agent (or, better yet, an agentless solution) that would simply know when data arrived. Then it could provision that data from the FTP server's data storage to the SQL database.

But that wasn't our only challenge. At the same time, our SQL and Oracle databases needed to remain in strong synchronization. Changes to specific values in SQL must replicate to Oracle, also in as real-time as computationally possible. Synchronizing Oracle with SQL meant also synchronizing metadata with SAP.

Even a single one of these near–real-time requirements can be challenging for a developer to build. Lacking developers, a development budget, and the desire to complete this project using off-the-shelf components, we demanded a solution that would accomplish the task without reverting to low-level coding.

What we needed were *triggers*.

Triggers are the real juice in an IT job scheduling solution. The kinds and capabilities of triggers a job scheduling solution supports makes the determination between one that's enterprise ready and one that's not much more than the Windows Task Scheduler.

The Project required a wide range of these triggers to get the job done: Our project's FTP-to-SQL integration required a *file-based trigger*, kicking off a job or plan when a file appeared at the FTP server. *Message-based triggers* were also necessary for the SQL-to-Oracle integration, enabling the two applications to notify each other about synchronization activities. *Event triggers* were necessary in the Oracle-to-SAP integration, allowing Oracle to create events about changes to its state and alerting SAP to make associated changes based on event characteristics. Those same event triggers gave Active Directory (AD) the data to quickly tag permissions into the data. Finally, *time-based triggers* kicked off occasional data transfers between the SQL database and the UNIX mainframe.

Wildly, simple triggers alone weren't sufficient. All by itself even the best trigger couldn't fulfill the multi-server and multi-action real-time requirement our system demanded. We also needed the ability to tether or "chain" individual triggers together. By chaining triggers, we could speed the process, get data where needed, and ensure the system remained in convergence. I'll talk more about chaining triggers in the next chapter.

Pay careful attention to the triggering capability of your selected IT job scheduling solution. The very best will come with the richest suite of triggering abilities.

Story #7: Securing the Data Center from the Ease of Script Execution

Company F was a midsize company with a midsize IT organization. Responsible for all the tasks typically associated with IT, its IT team got along well and generally provided good service to the company at large.

But even the most well-meaning of IT organizations occasionally makes mistakes, and sometimes those mistakes are large in impact. That's the situation that occurred one day after two administrators began sharing their quiver of scripts, tasks, and packages.

You already know that one of the primary benefits of moving to script-based automation is consistency in reuse. By packaging a series of tasks into a script, that script can be executed over and over with a known result. Parameterizing those same scripts makes them even more valuable to IT operations. Once created, the same script can be used over and over again across a range of different needs. Yet reusability can sometimes be a risk as well. When a person can simply double-click a script to activate it, the chance presents itself that that double-click might happen inappropriately.

That's exactly what was intended the day Sara "borrowed" one of Jane's scripts for use in another system. Jane's scripts were brilliantly designed, smartly parameterized, and well documented. Built right into each script was all the necessary information another administrator would need to reuse the script elsewhere. For Sara, Jane's script perfectly solved her problem at hand.

The central problem, however, was that Sara wasn't really authorized to run Jane's script. In fact, the well-meaning Sara wasn't supposed to be working on the system at all. When she executed Jane's script, it brought the system down unexpectedly. Company F learned a valuable lesson that day in the openness of simple scripts.

That's why shortly thereafter Company F invested in an IT job scheduling solution for aggregating their automations into a unified store. Unifying automations within a restricted-run framework enabled Company F to apply privileges to their scripts. Because they chose a best-in-class IT job scheduling solution, they were able to not only apply privileges on the scripts themselves but also the jobs, plans, and even variables associated with those scripts. Having correct permissions in place reduces the risk that a Sara will inappropriately execute a script. But, more importantly, it also reduces the risk that a Sara-type worker will inappropriately attach the wrong variables to the right script, or the right variables to the wrong script.

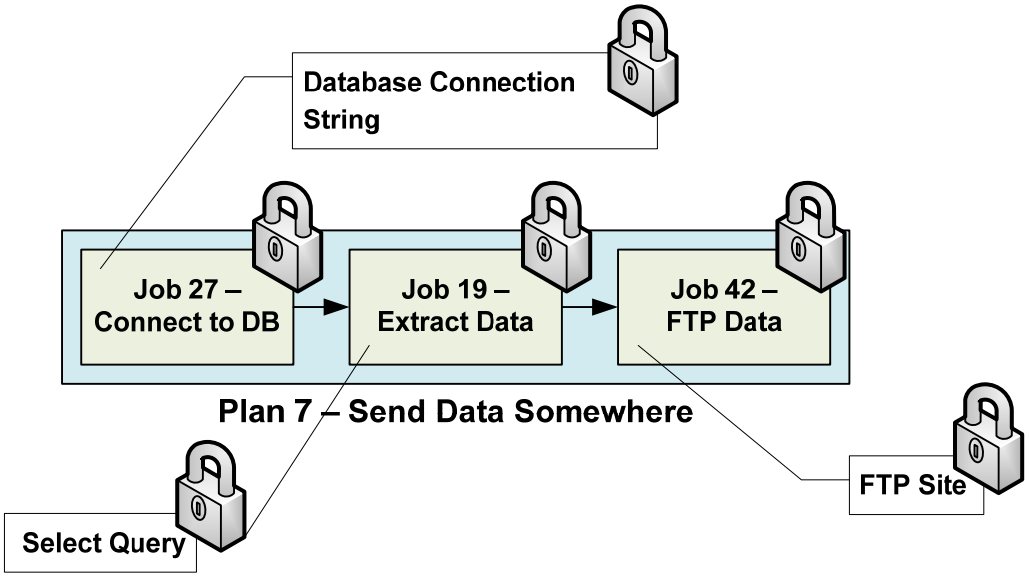

Figure 2.12: Applying security at various levels in script execution.

Consolidating IT automations into a job scheduling framework provides visualization of scripts within the enterprise. It adds security to what might otherwise be highly-dangerous text files. It creates a location where their successful execution can be proven to administrators as well as auditors. And it creates an auditable environment of approved execution that protects the data center.

Job Scheduling's Stories Are its Value Add

These seven stories are told to help you understand the value add to an IT job scheduling solution. In them, you've learned how IT job scheduling works for administration as well as complex tasks that might otherwise be relegated to low-level developers. You've discovered how triggers and file manipulations are as important as database tasks and middleware actions. You've also learned how job scheduling creates that framework for approved execution that your auditors—and, indeed, your entire business—truly appreciate.

Yet as I mentioned as this chapter began, I still haven't dug deeply into the inner workings of jobs themselves. Now that you've gained an appreciation for where job scheduling benefits the data center, let's spend some time discussing a technical deconstruction of an IT workflow. At the conclusion of the next chapter, you'll gain an even greater appreciation for how these job objects and their plans fit perfectly into the needs of your business services.